

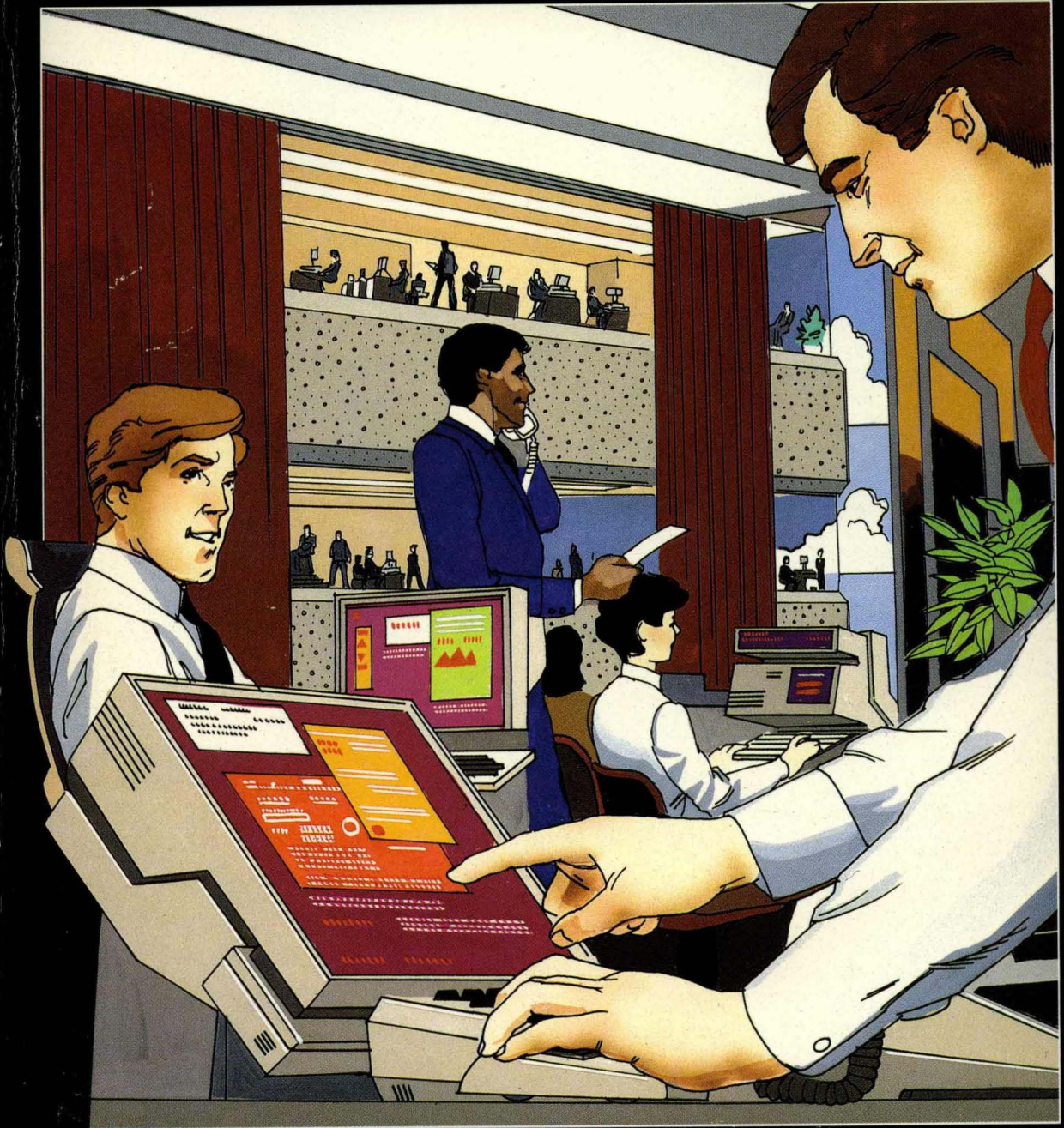
intel®

Microprocessors

1990

intel®

Microprocessors



Alvarado Peris/Scott

Order Number 230843-007



Intel the Microcomputer Company:

When Intel invented the microprocessor in 1971, it created the era of microcomputers. Whether used in embedded applications such as automobiles or microwave ovens, or as the CPU in personal computers or supercomputers, Intel's microcomputers have always offered leading-edge technology. Intel continues to strive for the highest standards in memory, microcomputer components, modules and systems to give its customers the best possible competitive advantages.

MICROPROCESSORS

1990



Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

376, 386, 387, 486, 4-SITE, Above, ACE51, ACE96, ACE186, ACE196, ACE960, BITBUS, COMMputer, CREDIT, Data Pipeline, ETOX, Genius, \hat{i} , i486, i860, ICE, iCEL, ICEVIEW, iCS, iDBP, iDIS, i \hat{I} CE, iLBX, iMDDX, iMMX, Inboard, Insite, Intel, int \hat{e} l, Intel386, int \hat{e} lBOS, Intel Certified, Intelevison, int \hat{e} l \hat{i} gent Identifier, int \hat{e} l \hat{i} gent Programming, Intellec, Intellink, iOSP, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iSXM, Library Manager, MAPNET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, MultiSERVER, ONCE, OpenNET, OTP, PROMPT, Promware, QUEST, QueX, Quick-Erase, Quick-Pulse Programming, Ripplemode, RMX/80, RUP1, Seamless, SLD, SugarCube, UPI, and VLSiCEL, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

CHMOS and HMOS are patented processes of Intel Corp.

Intel Corporation and Intel's FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641



CUSTOMER SUPPORT

INTEL'S COMPLETE SUPPORT SOLUTION WORLDWIDE

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, consulting services and network management services. For detailed information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is quite extensive. It can start with assistance during your development effort to network management. 100 Intel sales and service offices are located worldwide—in the U.S., Canada, Europe and the Far East. So wherever you're using Intel technology, our professional staff is within close reach.

HARDWARE SUPPORT SERVICES

Intel's hardware maintenance service, starting with complete on-site installation will boost your productivity from the start and keep you running at maximum efficiency. Support for system or board level products can be tailored to match your needs, from complete on-site repair and maintenance support to economical carry-in or mail-in factory service.

Intel can provide support service for not only Intel systems and emulators, but also support for equipment in your development lab or provide service on your product to your end-user/customer.

SOFTWARE SUPPORT SERVICES

Software products are supported by our Technical Information Service (TIPS) that has a special toll free number to provide you with direct, ready information on known, documented problems and deficiencies, as well as work-arounds, patches and other solutions.

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and *COMMENTS Magazine*). Basic support consists of updates and the subscription service. Contracts are sold in environments which represent product groupings (e.g., iRMX® environment).

CONSULTING SERVICES

Intel provides field system engineering consulting services for any phase of your development or application effort. You can use our system engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training and customizing an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, BITBUS™ and LAN applications.

NETWORK MANAGEMENT SERVICES

Today's networking products are powerful and extremely flexible. The return they can provide on your investment via increased productivity and reduced costs can be very substantial.

Intel offers complete network support, from definition of your network's physical and functional design, to implementation, installation and maintenance. Whether installing your first network or adding to an existing one, Intel's Networking Specialists can optimize network performance for you.

Table of Contents

Alphanumeric Index	ix
CHAPTER 1	
Overview	
Introduction	1-1
CHAPTER 2	
8086 Microprocessor Family	
DATA SHEETS	
8086 16-Bit HMOS Microprocessor 8086/8086-2/8086-1*	2-1
80C86A 16-Bit CHMOS Microprocessor	2-31
8088 8-Bit HMOS Microprocessor 8088/8088-2	2-60
80C88A 8-Bit CHMOS Microprocessor	2-90
8087 Math Coprocessor	2-122
CHAPTER 3	
80286 Microprocessor Family	
DATA SHEETS	
80C286 High Performance CHMOS Microprocessor with Memory Management and Protection	3-1
80286 High Performance Microprocessor with Memory Management and Protection	3-61
80C287A CHMOS III Math Coprocessor	3-118
80287 Math Coprocessor	3-119
82C288 Bus Controller for 80286 Processors (82C288-12, 82C288-10, 82C288-8) ..	3-145
82C284 Clock Generator and Ready Interface for 80286 Processors (82C284-12, 82C284-10, 82C284-8)	3-166
CHAPTER 4	
INTEL386™ Family	
DATA SHEETS	
i486 Microprocessor	4-1
82480 DMA Coprocessor	4-169
386 DX Microprocessor High Performance 32-Bit CHMOS Microprocessor with Integrated Memory Management	4-170
387 DX Math Coprocessor	4-305
82385 High Performance 32-Bit Cache Controller	4-343
386 SX Microprocessor	4-410
387 SX Math Coprocessor	4-507
82385SX High Performance Cache Controller	4-544
82380 High Performance 32-Bit DMA Controller with Integrated System Support Peripherals	4-618
376 High Performance 32-Bit Embedded Processor	4-752
82370 Integrated System Peripheral	4-843
CHAPTER 5	
i860™ Microprocessor Family	
i860 64-Bit Microprocessor	5-1
CHAPTER 6	
Development Tools for the 8086, 80186, 80188, 80286, 80386, and 80486	
LANGUAGES AND SOFTWARE DEVELOPMENT TOOLS	
8086/80186 Software Development Packages	6-1
iC-86/286 C Compiler	6-7
Ada: Cross-Development for the 80386 Microprocessor	6-10
Intel386 Family Development Support	6-14
AEDIT Source Code and Text Editor	6-18

Table of Contents (Continued)

iPAT Performance Analysis Tool	6-20
Validated Ada for UNIX/386	6-24
Ada-386/iRMK Interface Libraries	6-28
i486™ Microprocessor Development Tools	6-31
ICD-486 In-Circuit Debugger	6-35
IN-CIRCUIT EMULATORS	
I2ICE In-Circuit Emulation System	6-37
ICE-186 In-Circuit Emulator	6-40
ICE-188 In-Circuit Emulator	6-44
ICE-286 In-Circuit Emulator	6-48
Intel386™ Family In-Circuit Emulator	6-51
ICE-386 SX Specifications and Requirements	6-55
ICE-376 Specifications and Requirements	6-57
ICE-386 25MHz Specifications and Requirements	6-59

Alphanumeric Index

376 High Performance 32-Bit Embedded Processor	4-752
386 DX Microprocessor High Performance 32-Bit CHMOS Microprocessor with Integrated Memory Management	4-170
386 SX Microprocessor	4-410
387 DX Math Coprocessor	4-305
387 SX Math Coprocessor	4-507
80286 High Performance Microprocessor with Memory Management and Protection	3-61
80287 Math Coprocessor	3-119
8086 16-Bit HMOS Microprocessor 8086/8086-2/8086-1*	2-1
8086/80186 Software Development Packages	6-1
8087 Math Coprocessor	2-122
8088 8-Bit HMOS Microprocessor 8088/8088-2	2-60
80C286 High Performance CHMOS Microprocessor with Memory Management and Protection	3-1
80C287A CHMOS III Math Coprocessor	3-118
80C86A 16-Bit CHMOS Microprocessor	2-31
80C88A 8-Bit CHMOS Microprocessor	2-90
82370 Integrated System Peripheral	4-843
82380 High Performance 32-Bit DMA Controller with Integrated System Support Peripherals	4-618
82385 High Performance 32-Bit Cache Controller	4-343
82385SX High Performance Cache Controller	4-544
82480 DMA Coprocessor	4-169
82C284 Clock Generator and Ready Interface for 80286 Processors (82C284-12, 82C284-10, 82C284-8)	3-166
82C288 Bus Controller for 80286 Processors (82C288-12, 82C288-10, 82C288-8)	3-145
Ada-386/iRMK Interface Libraries	6-28
Ada: Cross-Development for the 80386 Microprocessor	6-10
AEDIT Source Code and Text Editor	6-18
I2ICE In-Circuit Emulation System	6-37
i486 Microprocessor	4-1
i486™ Microprocessor Development Tools	6-31
i860 64-Bit Microprocessor	5-1
iC-86/286 C Compiler	6-7
ICD-486 In-Circuit Debugger	6-35
ICE-186 In-Circuit Emulator	6-40
ICE-188 In-Circuit Emulator	6-44
ICE-286 In-Circuit Emulator	6-48
ICE-376 Specifications and Requirements	6-57
ICE-386 25MHz Specifications and Requirements	6-59
ICE-386 SX Specifications and Requirements	6-55
Intel386 Family Development Support	6-14
Intel386™ Family In-Circuit Emulator	6-51
iPAT Performance Analysis Tool	6-20
Validated Ada for UNIX/386	6-24

Overview

1

INTRODUCTION

Intel microprocessors and peripherals provide a complete solution in increasingly complex application environments. Quite often, a single peripheral device will replace anywhere from 20 to 100 TTL devices (and the associated design time that goes with them).

Built-in functions and standard Intel microprocessor/peripheral interface deliver very real *time* and *performance* advantages to the designer of microprocessor-based systems.

REDUCED TIME TO MARKET

When you can purchase an off-the-shelf solution that replaces a number of discrete devices, you're also replacing all the design, testing, and debug *time* that goes with them.

INCREASED RELIABILITY

At Intel, the rate of failure for devices is carefully tracked. Highest reliability is a tangible goal that translates to higher reliability for your product, reduced downtime, and reduced

repair costs. And as more and more functions are integrated on a single VLSI device, the resulting system requires less power, produces less heat, and requires fewer mechanical connections — again resulting in greater system reliability.

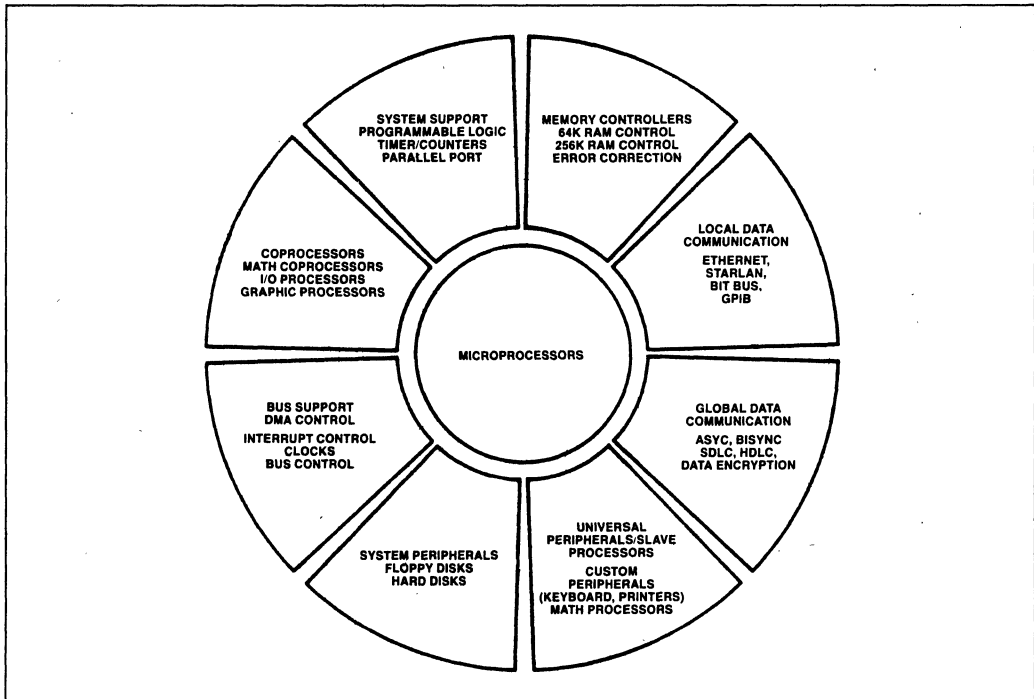
LOWER PRODUCTION COST

By minimizing design time, increasing reliability, and replacing numerous parts, microprocessor and peripheral solutions can contribute dramatically to lower product costs.

HIGHER SYSTEM PERFORMANCE

Intel microprocessors and peripherals provide the highest system performance for the demands of today's (and tomorrow's) microprocessor-based applications. For example, the Intel386™ Microprocessor Family offers 32-bit performance for multitasking, multiuser systems. Intel's peripheral products have been designed with the future in mind. They support all of Intel's 8, 16 and 32 bit processors.

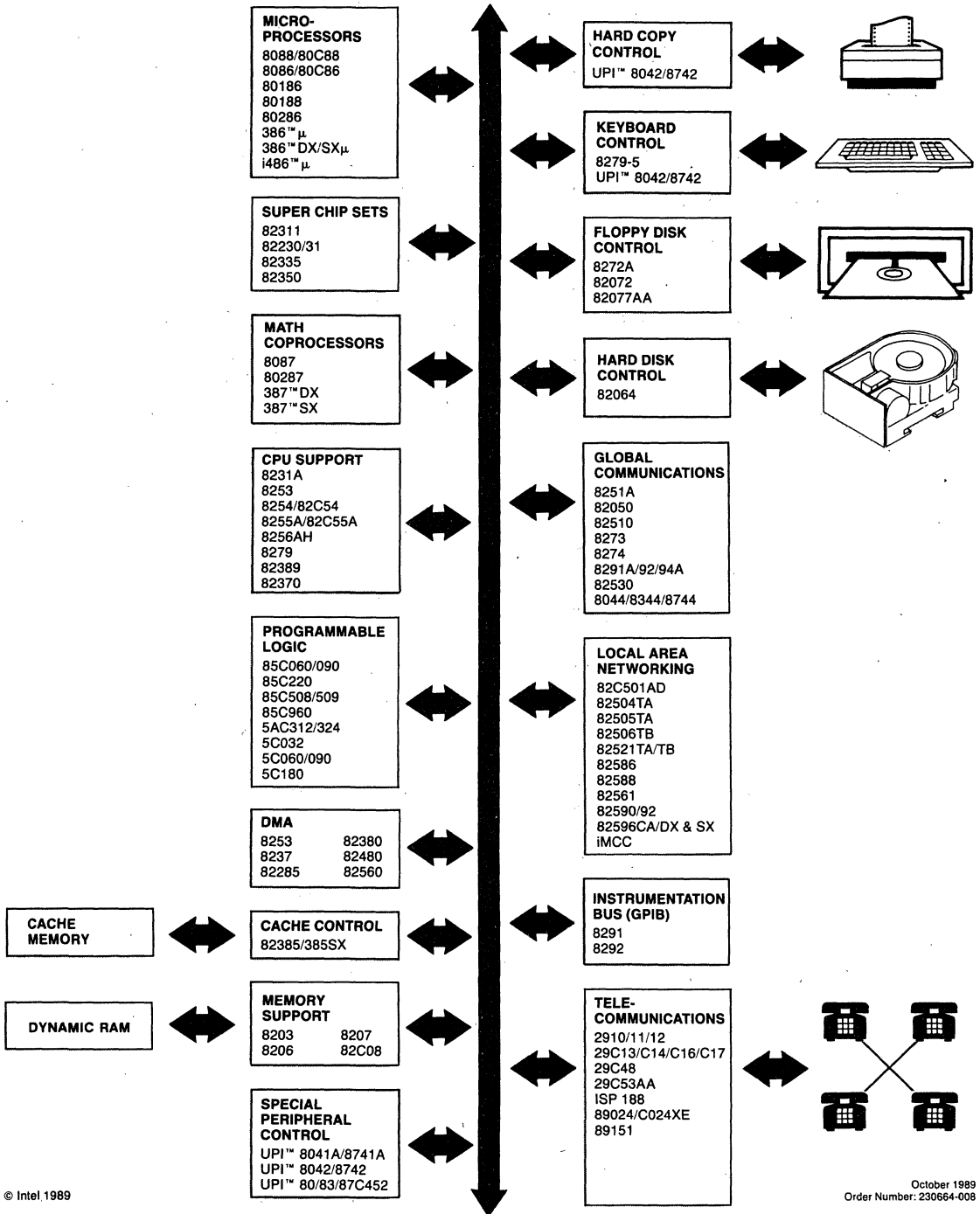
The Intel microprocessor and peripherals family provides a broad range of time-saving, high performance solutions.





Get Your Kit Together!

Intel's Microsystem Components Kit Solution





8086

16-BIT HMOS MICROPROCESSOR

8086/8086-2/8086-1*

- Direct Addressing Capability 1 MByte of Memory
- Architecture Designed for Powerful Assembly Language and Efficient High Level Languages
- 14 Word, by 16-Bit Register Set with Symmetrical Operations
- 24 Operand Addressing Modes
- Bit, Byte, Word, and Block Operations
- 8 and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal Including Multiply and Divide
- Range of Clock Rates:
 - 5 MHz for 8086,
 - 8 MHz for 8086-2,
 - 10 MHz for 8086-1
- MULTIBUS® System Compatible Interface
- Available in EXPRESS
 - Standard Temperature Range
 - Extended Temperature Range
- Available in 40-Lead Cerdip and Plastic Package
 - (See Packaging Spec. Order # 231369)

The Intel 8086 high performance 16-bit CPU is available in three clock rates: 5, 8 and 10 MHz. The CPU is implemented in N-Channel, depletion load, silicon gate technology (HMOS), and packaged in a 40-pin CERDIP or plastic package. The 8086 operates in both single processor and multiple processor configurations to achieve high performance levels.

*Changes from the 1985 handbook specification have been made for the 8086-1. See A.C. Characteristics TGvCH and TCLGL.

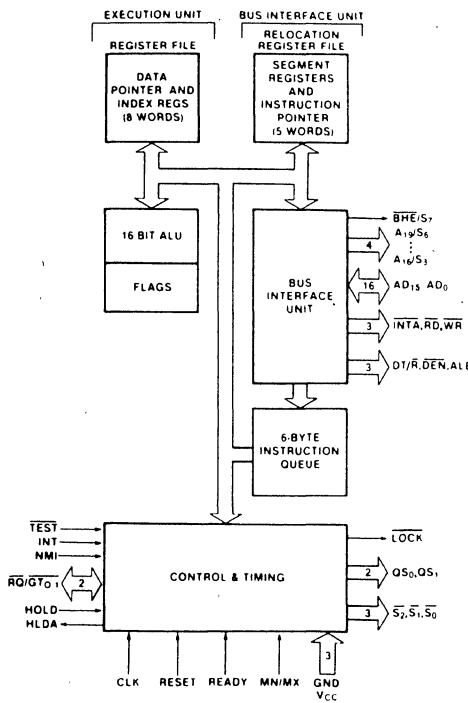
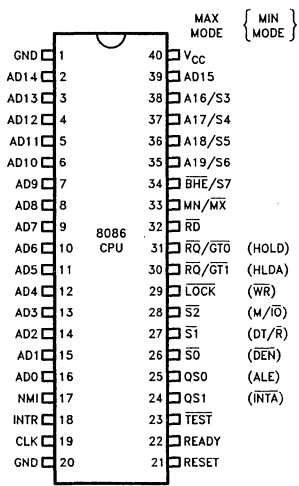


Figure 1. 8086 CPU Block Diagram

231455-1



40 Lead
Figure 2. 8086 Pin Configuration

231455-2

Table 1. Pin Description

The following pin function descriptions are for 8086 systems in either minimum or maximum mode. The "Local Bus" in these descriptions is the direct multiplexed bus interface connection to the 8086 (without regard to additional bus buffers).

Symbol	Pin No.	Type	Name and Function																		
AD ₁₅ -AD ₀	2-16, 39	I/O	<p>ADDRESS DATA BUS: These lines constitute the time multiplexed memory/I/O address (T₁), and data (T₂, T₃, T_W, T₄) bus. A₀ is analogous to BHE for the lower byte of the data bus, pins D₇-D₀. It is LOW during T₁ when a byte is to be transferred on the lower portion of the bus in memory or I/O operations. Eight-bit oriented devices tied to the lower half would normally use A₀ to condition chip select functions. (See BHE.) These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".</p>																		
A ₁₉ /S ₆ , A ₁₈ /S ₅ , A ₁₇ /S ₄ , A ₁₆ /S ₃	35-38	O	<p>ADDRESS/STATUS: During T₁ these are the four most significant address lines for memory operations. During I/O operations these lines are LOW. During memory and I/O operations, status information is available on these lines during T₂, T₃, T_W, T₄. The status of the interrupt enable FLAG bit (S₅) is updated at the beginning of each CLK cycle. A₁₇/S₄ and A₁₆/S₃ are encoded as shown. This information indicates which relocation register is presently being used for data accessing. These lines float to 3-state OFF during local bus "hold acknowledge."</p>																		
			<table border="1"> <thead> <tr> <th>A₁₇/S₄</th> <th>A₁₆/S₃</th> <th>Characteristics</th> </tr> </thead> <tbody> <tr> <td>0 (LOW)</td> <td>0</td> <td>Alternate Data</td> </tr> <tr> <td>0</td> <td>1</td> <td>Stack</td> </tr> <tr> <td>1 (HIGH)</td> <td>0</td> <td>Code or None</td> </tr> <tr> <td>1</td> <td>1</td> <td>Data</td> </tr> <tr> <td>S₆ is 0 (LOW)</td> <td></td> <td></td> </tr> </tbody> </table>	A ₁₇ /S ₄	A ₁₆ /S ₃	Characteristics	0 (LOW)	0	Alternate Data	0	1	Stack	1 (HIGH)	0	Code or None	1	1	Data	S ₆ is 0 (LOW)		
			A ₁₇ /S ₄	A ₁₆ /S ₃	Characteristics																
0 (LOW)	0	Alternate Data																			
0	1	Stack																			
1 (HIGH)	0	Code or None																			
1	1	Data																			
S ₆ is 0 (LOW)																					
BHE/S ₇	34	O	<p>BUS HIGH ENABLE/STATUS: During T₁ the bus high enable signal (BHE) should be used to enable data onto the most significant half of the data bus, pins D₁₅-D₈. Eight-bit oriented devices tied to the upper half of the bus would normally use BHE to condition chip select functions. BHE is LOW during T₁ for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. The S₇ status information is available during T₂, T₃, and T₄. The signal is active LOW, and floats to 3-state OFF in "hold". It is LOW during T₁ for the first interrupt acknowledge cycle.</p>																		
			<table border="1"> <thead> <tr> <th>BHE</th> <th>A₀</th> <th>Characteristics</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Whole word</td> </tr> <tr> <td>0</td> <td>1</td> <td>Upper byte from/to odd address</td> </tr> <tr> <td>1</td> <td>0</td> <td>Lower byte from/to even address</td> </tr> <tr> <td>1</td> <td>1</td> <td>None</td> </tr> </tbody> </table>	BHE	A ₀	Characteristics	0	0	Whole word	0	1	Upper byte from/to odd address	1	0	Lower byte from/to even address	1	1	None			
			BHE	A ₀	Characteristics																
0	0	Whole word																			
0	1	Upper byte from/to odd address																			
1	0	Lower byte from/to even address																			
1	1	None																			
RD	32	O	<p>READ: Read strobe indicates that the processor is performing a memory or I/O read cycle, depending on the state of the S₂ pin. This signal is used to read devices which reside on the 8086 local bus. RD is active LOW during T₂, T₃ and T_W of any read cycle, and is guaranteed to remain HIGH in T₂ until the 8086 local bus has floated. This signal floats to 3-state OFF in "hold acknowledge".</p>																		

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
READY	22	I	READY: is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory/I/O is synchronized by the 8284A Clock Generator to form READY. This signal is active HIGH. The 8086 READY input is not synchronized. Correct operation is not guaranteed if the setup and hold times are not met.
INTR	18	I	INTERRUPT REQUEST: is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.
$\overline{\text{TEST}}$	23	I	$\overline{\text{TEST}}$: input is examined by the "Wait" instruction. If the $\overline{\text{TEST}}$ input is LOW execution continues, otherwise the processor waits in an "Idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.
NMI	17	I	NON-MASKABLE INTERRUPT: an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.
RESET	21	I	RESET: causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the Instruction Set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	CLOCK: provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V _{CC}	40		V_{CC}: +5V power supply pin.
GND	1, 20		GROUND
MN/ $\overline{\text{MX}}$	33	I	MINIMUM/MAXIMUM: indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

The following pin function descriptions are for the 8086/8288 system in maximum mode (i.e., $\text{MN}/\overline{\text{MX}} = V_{SS}$). Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

$\overline{\text{S}}_2, \overline{\text{S}}_1, \overline{\text{S}}_0$	26-28	O	STATUS: active during T_4 , T_1 , and T_2 and is returned to the passive state (1, 1, 1) during T_3 or during T_W when READY is HIGH. This status is used by the 8288 Bus Controller to generate all memory and I/O access control signals. Any change by $\overline{\text{S}}_2$, $\overline{\text{S}}_1$, or $\overline{\text{S}}_0$ during T_4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle.
---	-------	---	--

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function			
$\overline{S_2}, \overline{S_1}, \overline{S_0}$ (Continued)	26-28	O	These signals float to 3-state OFF in "hold acknowledge". These status lines are encoded as shown.			
			$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Characteristics
			0 (LOW)	0	0	Interrupt Acknowledge
			0	0	1	Read I/O Port
			0	1	0	Write I/O Port
0	1	1	Halt			
1 (HIGH)	0	0	Code Access			
1	0	1	Read Memory			
1	1	0	Write Memory			
1	1	1	Passive			
$\overline{RQ}/\overline{GT_0}$, $\overline{RQ}/\overline{GT_1}$	30, 31	I/O	<p>REQUEST/GRANT: pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT_0}$ having higher priority than $\overline{RQ}/\overline{GT_1}$. $\overline{RQ}/\overline{GT}$ pins have internal pull-up resistors and may be left unconnected. The request/grant sequence is as follows (see Figure 9):</p> <ol style="list-style-type: none"> 1. A pulse of 1 CLK wide from another local bus master indicates a local bus request ("hold") to the 8086 (pulse 1). 2. During a T_4 or T_1 clock cycle, a pulse 1 CLK wide from the 8086 to the requesting master (pulse 2), indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge". 3. A pulse 1 CLK wide from the requesting master indicates to the 8086 (pulse 3) that the "hold" request is about to end and that the 8086 can reclaim the local bus at the next CLK. <p>Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW.</p> <p>If the request is made while the CPU is performing a memory cycle, it will release the local bus during T_4 of the cycle when all the following conditions are met:</p> <ol style="list-style-type: none"> 1. Request occurs on or before T_2. 2. Current cycle is not the low byte of a word (on an odd address). 3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence. 4. A locked instruction is not currently executing. <p>If the local bus is idle when the request is made the two possible events will follow:</p> <ol style="list-style-type: none"> 1. Local bus will be released during the next clock. 2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied. 			
LOCK	29	O	<p>LOCK: output indicates that other system bus masters are not to gain control of the system bus while LOCK is active LOW. The LOCK signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state OFF in "hold acknowledge".</p>			

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function		
QS ₁ , QS ₀	24, 25	O	QUEUE STATUS: The queue status is valid during the CLK cycle after which the queue operation is performed. QS ₁ and QS ₀ provide status to allow external tracking of the internal 8086 instruction queue.		
			QS ₁	QS ₀	Characteristics
			0 (LOW) 0 1 (HIGH) 1	0 1 0 1	No Operation First Byte of Op Code from Queue Empty the Queue Subsequent Byte from Queue

The following pin function descriptions are for the 8086 in minimum mode (i.e., $MN/\overline{MX} = V_{CC}$). Only the pin functions which are unique to minimum mode are described; all other pin functions are as described above.

$\overline{M/\overline{IO}}$	28	O	STATUS LINE: logically equivalent to S ₂ in the maximum mode. It is used to distinguish a memory access from an I/O access. $\overline{M/\overline{IO}}$ becomes valid in the T ₄ preceding a bus cycle and remains valid until the final T ₄ of the cycle (M = HIGH, IO = LOW). $\overline{M/\overline{IO}}$ floats to 3-state OFF in local bus "hold acknowledge".
\overline{WR}	29	O	WRITE: indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the $\overline{M/\overline{IO}}$ signal. \overline{WR} is active for T ₂ , T ₃ and T _W of any write cycle. It is active LOW, and floats to 3-state OFF in local bus "hold acknowledge".
\overline{INTA}	24	O	\overline{INTA}: is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T ₂ , T ₃ and T _W of each interrupt acknowledge cycle.
ALE	25	O	ADDRESS LATCH ENABLE: provided by the processor to latch the address into the 8282/8283 address latch. It is a HIGH pulse active during T ₁ of any bus cycle. Note that ALE is never floated.
$\overline{DT/\overline{R}}$	27	O	DATA TRANSMIT/RECEIVE: needed in minimum system that desires to use an 8286/8287 data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically $\overline{DT/\overline{R}}$ is equivalent to S ₁ in the maximum mode, and its timing is the same as for $\overline{M/\overline{IO}}$. (T = HIGH, R = LOW.) This signal floats to 3-state OFF in local bus "hold acknowledge".
\overline{DEN}	26	O	DATA ENABLE: provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. \overline{DEN} is active LOW during each memory and I/O access and for \overline{INTA} cycles. For a read or \overline{INTA} cycle it is active from the middle of T ₂ until the middle of T ₄ , while for a write cycle it is active from the beginning of T ₂ until the middle of T ₄ . \overline{DEN} floats to 3-state OFF in local bus "hold acknowledge".
HOLD, HLDA	31, 30	I/O	HOLD: indicates that another master is requesting a local bus "hold." To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement in the middle of a T ₄ or T ₁ clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor will LOWER the HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. Hold acknowledge (HLDA) and HOLD have internal pull-up resistors. The same rules as for $\overline{RQ/\overline{GT}}$ apply regarding when the local bus will be released. HOLD is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the setup time.

FUNCTIONAL DESCRIPTION

General Operation

The internal functions of the 8086 processor are partitioned logically into two processing units. The first is the Bus Interface Unit (BIU) and the second is the Execution Unit (EU) as shown in the block diagram of Figure 1.

These units can interact directly but for the most part perform as separate asynchronous operational processors. The bus interface unit provides the functions related to instruction fetching and queuing, operand fetch and store, and address relocation. This unit also provides the basic bus control. The overlap of instruction pre-fetching provided by this unit serves to increase processor performance through improved bus bandwidth utilization. Up to 6 bytes of the instruction stream can be queued while waiting for decoding and execution.

The instruction stream queuing mechanism allows the BIU to keep the memory utilized very efficiently. Whenever there is space for at least 2 bytes in the queue, the BIU will attempt a word fetch memory cycle. This greatly reduces "dead time" on the memory bus. The queue acts as a First-In-First-Out (FIFO) buffer, from which the EU extracts instruction bytes as required. If the queue is empty (following a branch instruction, for example), the first byte into the queue immediately becomes available to the EU.

The execution unit receives pre-fetched instructions from the BIU queue and provides un-relocated operand addresses to the BIU. Memory operands are passed through the BIU for processing by the EU, which passes results to the BIU for storage. See the Instruction Set description for further register set and architectural descriptions.

MEMORY ORGANIZATION

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million

bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64K bytes each, with each segment falling on 16-byte boundaries. (See Figure 3a.)

All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the addressing needs of programs. The segment register to be selected is automatically chosen according to the rules of the following table. All information in one segment type share the same logical attributes (e.g. code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster, and more structured.

Word (16-bit) operands can be located on even or odd address boundaries and are thus not constrained to even boundaries as is the case in many 16-bit computers. For address and data operands, the least significant byte of the word is stored in the lower valued address location and the most significant byte in the next higher address location. The BIU automatically performs the proper number of memory accesses, one if the word operand is on an even byte boundary and two if it is on an odd byte boundary. Except for the performance penalty, this double access is transparent to the software. This performance penalty does not occur for instruction fetches, only word operands.

Physically, the memory is organized as a high bank (D₁₅-D₈) and a low bank (D₇-D₀) of 512K 8-bit bytes addressed in parallel by the processor's address lines A₁₉-A₁. Byte data with even addresses is transferred on the D₇-D₀ bus lines while odd addressed byte data (A₀ HIGH) is transferred on the D₁₅-D₈ bus lines. The processor provides two enable signals, \overline{BHE} and A₀, to selectively allow reading from or writing into either an odd byte location, even byte location, or both. The instruction stream is fetched from memory as words and is addressed internally by the processor to the byte level as necessary.

Memory Reference Need	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch.
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
Local Data	DATA (DS)	Data references when: relative to stack, destination of string operation, or explicitly overridden.
External (Global) Data	EXTRA (ES)	Destination of string operations: explicitly selected using a segment override.

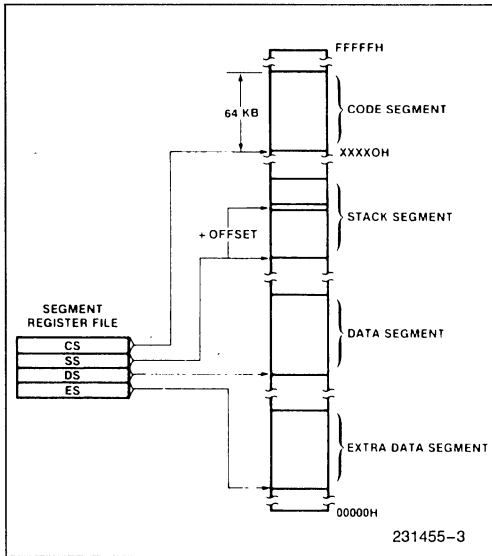


Figure 3a. Memory Organization

In referencing word data the BIU requires one or two memory cycles depending on whether or not the starting byte of the word is on an even or odd address, respectively. Consequently, in referencing word operands performance can be optimized by locating data on even address boundaries. This is an especially useful technique for using the stack, since odd address references to the stack may adversely affect the context switching time for interrupt processing or task multiplexing.

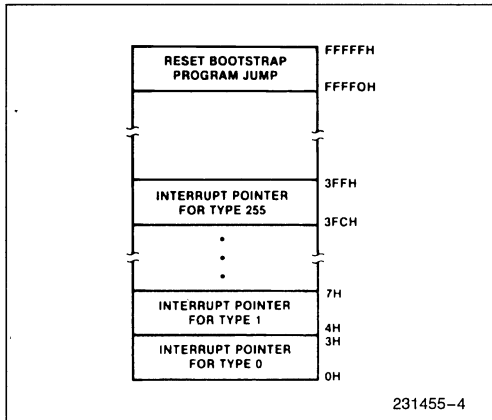


Figure 3b. Reserved Memory Locations

Certain locations in memory are reserved for specific CPU operations (see Figure 3b). Locations from

address FFFF0H through FFFFFH are reserved for operations including a jump to the initial program loading routine. Following RESET, the CPU will always begin execution at location FFFF0H where the jump must be. Locations 00000H through 003FFH are reserved for interrupt operations. Each of the 256 possible interrupt types has its service routine pointed to by a 4-byte pointer element consisting of a 16-bit segment address and a 16-bit offset address. The pointer elements are assumed to have been stored at the respective places in reserved memory prior to occurrence of interrupts.

MINIMUM AND MAXIMUM MODES

The requirements for supporting minimum and maximum 8086 systems are sufficiently different that they cannot be done efficiently with 40 uniquely defined pins. Consequently, the 8086 is equipped with a strap pin (MN/MX) which defines the system configuration. The definition of a certain subset of the pins changes dependent on the condition of the strap pin. When MN/MX pin is strapped to GND, the 8086 treats pins 24 through 31 in maximum mode. An 8288 bus controller interprets status information coded into $\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$ to generate bus timing and control signals compatible with the MULTIBUS® architecture. When the MN/MX pin is strapped to V_{CC} , the 8086 generates bus control signals itself on pins 24 through 31, as shown in parentheses in Figure 2. Examples of minimum mode and maximum mode systems are shown in Figure 4.

BUS OPERATION

The 8086 has a combined address and data bus commonly referred to as a time multiplexed bus. This technique provides the most efficient use of pins on the processor while permitting the use of a standard 40-lead package. This "local bus" can be buffered directly and used throughout the system with address latching provided on memory and I/O modules. In addition, the bus can also be demultiplexed at the processor with a single set of address latches if a standard non-multiplexed bus is desired for the system.

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T_1 , T_2 , T_3 and T_4 (see Figure 5). The address is emitted from the processor during T_1 and data transfer occurs on the bus during T_3 and T_4 . T_2 is used primarily for changing the direction of the bus during read operations. In the event that a "NOT READY" indication is given by the addressed device, "Wait" states (T_W) are inserted between T_3 and T_4 . Each inserted "Wait" state is of the same duration as a CLK cycle. Periods

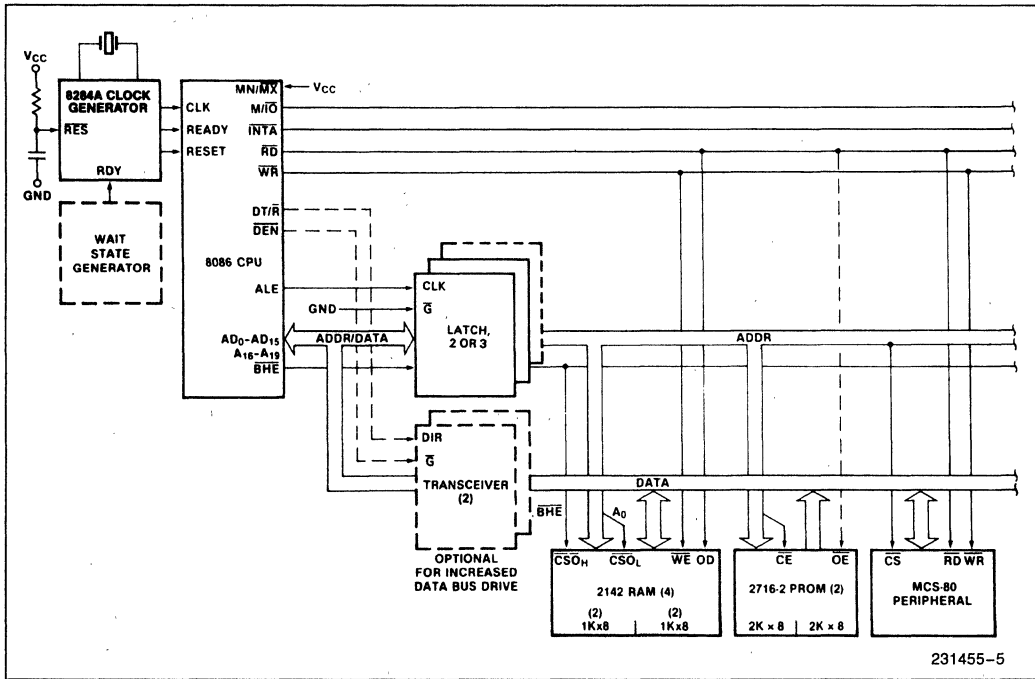


Figure 4a. Minimum Mode 8086 Typical Configuration

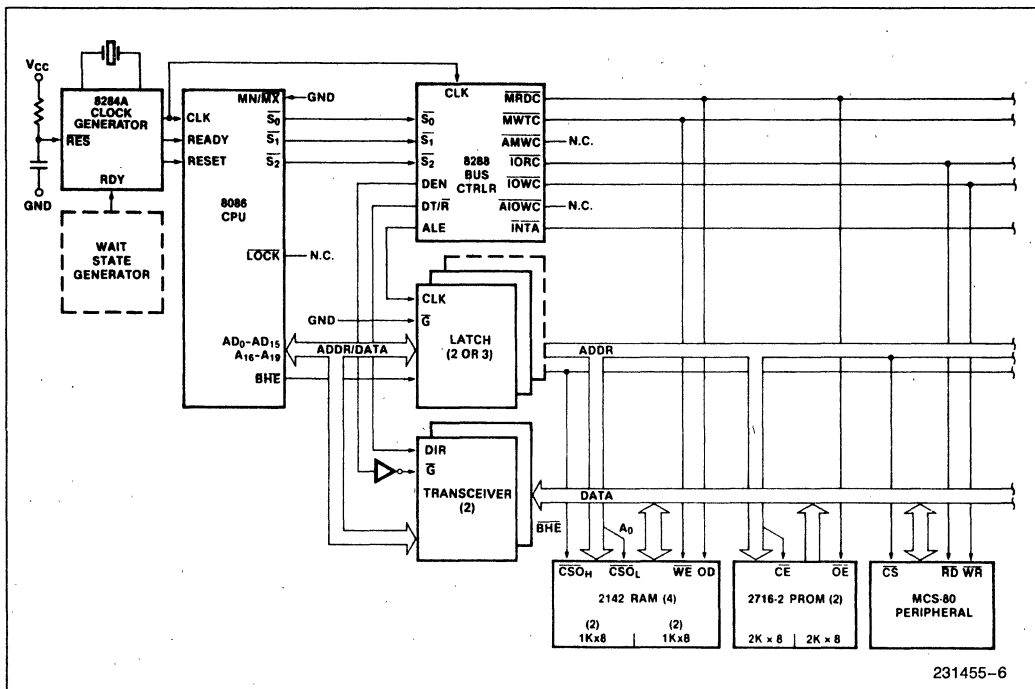


Figure 4b. Maximum Mode 8086 Typical Configuration

can occur between 8086 bus cycles. These are referred to as "Idle" states (T_i) or inactive CLK cycles. The processor uses these cycles for internal house-keeping.

During T_1 of any bus cycle the ALE (Address Latch Enable) signal is emitted (by either the processor or the 8288 bus controller, depending on the MN/ \overline{MX} strap). At the trailing edge of this pulse, a valid address and certain status information for the cycle may be latched.

Status bits $\overline{S_0}$, $\overline{S_1}$, and $\overline{S_2}$ are used, in maximum mode, by the bus controller to identify the type of bus transaction according to the following table:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Characteristics
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

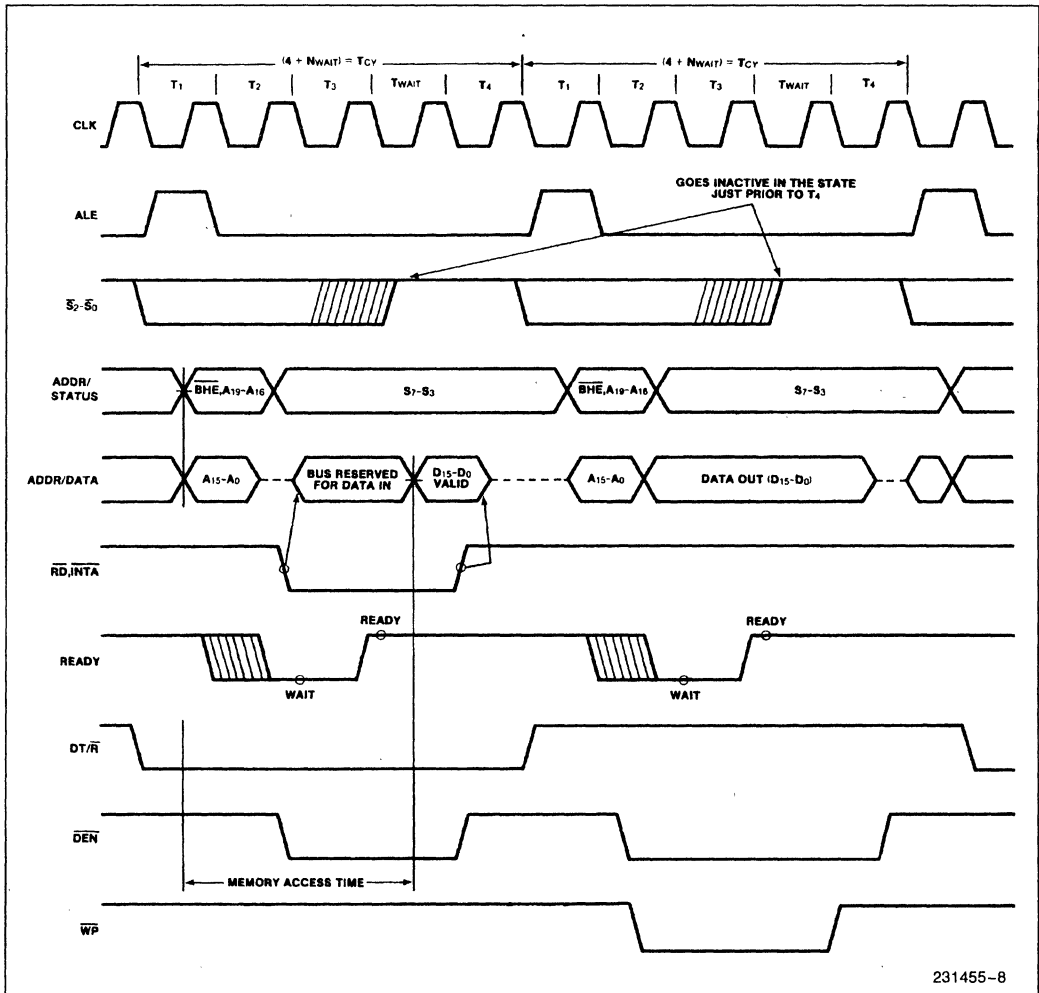


Figure 5. Basic System Timing

Status bits S_3 through S_7 are multiplexed with high-order address bits and the BHE signal, and are therefore valid during T_2 through T_4 . S_3 and S_4 indicate which segment register (see Instruction Set description) was used for this bus cycle in forming the address, according to the following table:

S_4	S_3	Characteristics
0 (LOW)	0	Alternate Data (extra segment)
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data

S_5 is a reflection of the PSW interrupt enable bit. $S_6 = 0$ and S_7 is a spare status bit.

I/O ADDRESSING

In the 8086, I/O operations can address up to a maximum of 64K I/O byte registers or 32K I/O word registers. The I/O address appears in the same format as the memory address on bus lines A_{15} – A_0 . The address lines A_{19} – A_{16} are zero in I/O operations. The variable I/O instructions which use register DX as a pointer have full address capability while the direct I/O instructions directly address one or two of the 256 I/O byte locations in page 0 of the I/O address space.

I/O ports are addressed in the same manner as memory locations. Even addressed bytes are transferred on the D_7 – D_0 bus lines and odd addressed bytes on D_{15} – D_8 . Care must be taken to assure that each register within an 8-bit peripheral located on the lower portion of the bus be addressed as even.

External Interface

PROCESSOR RESET AND INITIALIZATION

Processor initialization or start up is accomplished with activation (HIGH) of the RESET pin. The 8086 RESET is required to be HIGH for greater than 4 CLK cycles. The 8086 will terminate operations on the high-going edge of RESET and will remain dormant as long as RESET is HIGH. The low-going transition of RESET triggers an internal reset sequence for approximately 10 CLK cycles. After this interval the 8086 operates normally beginning with the instruction in absolute location $FFFF0H$ (see Figure 3b). The details of this operation are specified in the Instruction Set description of the MCS-86 Family User's Manual. The RESET input is internally synchronized to the processor clock. At initialization the HIGH-to-LOW transition of RESET must occur no sooner than 50 μs after power-up, to allow complete initialization of the 8086.

NMI asserted prior to the 2nd clock after the end of RESET will not be honored. If NMI is asserted after that point and during the internal reset sequence, the processor may execute one instruction before responding to the interrupt. A hold request active immediately after RESET will be honored before the first instruction fetch.

All 3-state outputs float to 3-state OFF during RESET. Status is active in the idle state for the first clock after RESET becomes active and then floats to 3-state OFF. ALE and HLDA are driven low.

INTERRUPT OPERATIONS

Interrupt operations fall into two classes; software or hardware initiated. The software initiated interrupts and software aspects of hardware interrupts are specified in the Instruction Set description. Hardware interrupts can be classified as non-maskable or maskable.

Interrupts result in a transfer of control to a new program location. A 256-element table containing address pointers to the interrupt service program locations resides in absolute locations 0 through 3FFH (see Figure 3b), which are reserved for this purpose. Each element in the table is 4 bytes in size and corresponds to an interrupt "type". An interrupting device supplies an 8-bit type number, during the interrupt acknowledge sequence, which is used to "vector" through the appropriate element to the new interrupt service program location.

NON-MASKABLE INTERRUPT (NMI)

The processor provides a single non-maskable interrupt pin (NMI) which has higher priority than the maskable interrupt request pin (INTR). A typical use would be to activate a power failure routine. The NMI is edge-triggered on a LOW-to-HIGH transition. The activation of this pin causes a type 2 interrupt. (See Instruction Set description.)

NMI is required to have a duration in the HIGH state of greater than two CLK cycles, but is not required to be synchronized to the clock. Any high-going transition of NMI is latched on-chip and will be serviced at the end of the current instruction or between whole moves of a block-type instruction. Worst case response to NMI would be for multiply, divide, and variable shift instructions. There is no specification on the occurrence of the low-going edge; it may occur before, during, or after the servicing of NMI. Another high-going edge triggers another response if it occurs after the start of the NMI procedure. The signal must be free of logical spikes in general and be free of bounces on the low-going edge to avoid triggering extraneous responses.

MASKABLE INTERRUPT (INTR)

The 8086 provides a single interrupt request input (INTR) which can be masked internally by software with the resetting of the interrupt enable FLAG status bit. The interrupt request signal is level triggered. It is internally synchronized during each clock cycle on the high-going edge of CLK. To be responded to, INTR must be present (HIGH) during the clock period preceding the end of the current instruction or the end of a whole move for a block-type instruction. During the interrupt response sequence further interrupts are disabled. The enable bit is reset as part of the response to any interrupt (INTR, NMI, software interrupt or single-step), although the FLAGS register which is automatically pushed onto the stack reflects the state of the processor prior to the interrupt. Until the old FLAGS register is restored the enable bit will be zero unless specifically set by an instruction.

During the response sequence (Figure 6) the processor executes two successive (back-to-back) interrupt acknowledge cycles. The 8086 emits the LOCK signal from T₂ of the first bus cycle until T₂ of the second. A local bus "hold" request will not be honored until the end of the second bus cycle. In the second bus cycle a byte is fetched from the external interrupt system (e.g., 8259A PIC) which identifies the source (type) of the interrupt. This byte is multiplied by four and used as a pointer into the interrupt vector lookup table. An INTR signal left HIGH will be continually responded to within the limitations of the enable bit and sample period. The INTERRUPT RETURN instruction includes a FLAGS pop which returns the status of the original interrupt enable bit when it restores the FLAGS.

HALT

When a software "HALT" instruction is executed the processor indicates that it is entering the "HALT" state in one of two ways depending upon which mode is strapped. In minimum mode, the processor issues one ALE with no qualifying bus control signals. In maximum mode, the processor issues appropriate HALT status on \overline{S}_2 , \overline{S}_1 , and \overline{S}_0 ; and the 8288 bus controller issues one ALE. The 8086 will not leave the "HALT" state when a local bus "hold" is entered while in "HALT". In this case, the processor reissues the HALT indicator. An interrupt request or RESET will force the 8086 out of the "HALT" state.

READ/MODIFY/WRITE (SEMAPHORE) OPERATIONS VIA LOCK

The \overline{LOCK} status information is provided by the processor when directly consecutive bus cycles are required during the execution of an instruction. This provides the processor with the capability of performing read/modify/write operations on memory (via the Exchange Register With Memory instruction, for example) without the possibility of another system bus master receiving intervening memory cycles. This is useful in multi-processor system configurations to accomplish "test and set lock" operations. The \overline{LOCK} signal is activated (forced LOW) in the clock cycle following the one in which the software "LOCK" prefix instruction is decoded by the EU. It is deactivated at the end of the last bus cycle of the instruction following the "LOCK" prefix instruction. While \overline{LOCK} is active a request on a RQ/GT pin will be recorded and then honored at the end of the LOCK.

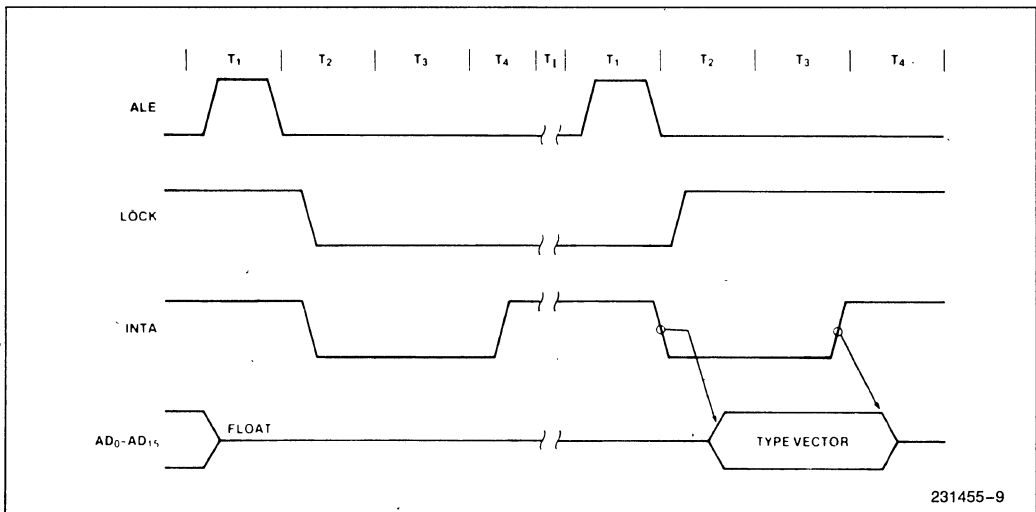


Figure 6. Interrupt Acknowledge Sequence

EXTERNAL SYNCHRONIZATION VIA TEST

As an alternative to the interrupts and general I/O capabilities, the 8086 provides a single software-testable input known as the TEST signal. At any time the program may execute a WAIT instruction. If at that time the TEST signal is inactive (HIGH), program execution becomes suspended while the processor waits for TEST to become active. It must remain active for at least 5 CLK cycles. The WAIT instruction is re-executed repeatedly until that time. This activity does not consume bus cycles. The processor remains in an idle state while waiting. All 8086 drivers go to 3-state OFF if bus "Hold" is entered. If interrupts are enabled, they may occur while the processor is waiting. When this occurs the processor fetches the WAIT instruction one extra time, processes the interrupt, and then re-fetches and re-executes the WAIT instruction upon returning from the interrupt.

Basic System Timing

Typical system configurations for the processor operating in minimum mode and in maximum mode are shown in Figures 4a and 4b, respectively. In minimum mode, the MN/MX pin is strapped to V_{CC} and the processor emits bus control signals in a manner similar to the 8085. In maximum mode, the MN/MX pin is strapped to V_{SS} and the processor emits coded status information which the 8288 bus controller uses to generate MULTIBUS compatible bus control signals. Figure 5 illustrates the signal timing relationships.

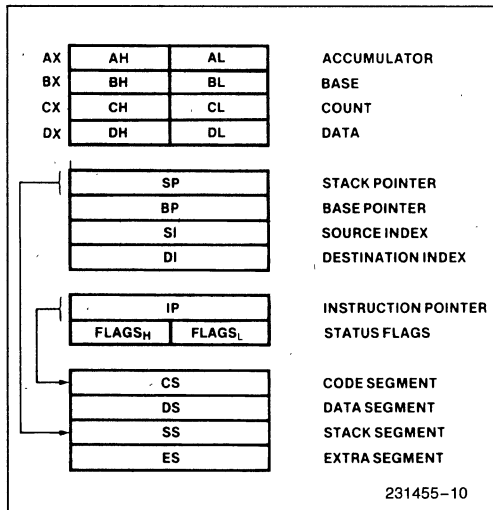


Figure 7. 8086 Register Model

SYSTEM TIMING—MINIMUM SYSTEM

The read cycle begins in T_1 with the assertion of the Address Latch Enable (ALE) signal. The trailing (low-going) edge of this signal is used to latch the address information, which is valid on the local bus at this time, into the address latch. The \overline{BHE} and A_0 signals address the low, high, or both bytes. From T_1 to T_4 the M/\overline{IO} signal indicates a memory or I/O operation. At T_2 the address is removed from the local bus and the bus goes to a high impedance state. The read control signal is also asserted at T_2 . The read (\overline{RD}) signal causes the addressed device to enable its data bus drivers to the local bus. Some time later valid data will be available on the bus and the addressed device will drive the READY line HIGH. When the processor returns the read signal to a HIGH level, the addressed device will again 3-state its bus drivers. If a transceiver is required to buffer the 8086 local bus, signals $\overline{DT/R}$ and \overline{DEN} are provided by the 8086.

A write cycle also begins with the assertion of ALE and the emission of the address. The M/\overline{IO} signal is again asserted to indicate a memory or I/O write operation. In the T_2 immediately following the address emission the processor emits the data to be written into the addressed location. This data remains valid until the middle of T_4 . During T_2 , T_3 , and T_4 the processor asserts the write control signal. The write (\overline{WR}) signal becomes active at the beginning of T_2 as opposed to the read which is delayed somewhat into T_2 to provide time for the bus to float.

The \overline{BHE} and A_0 signals are used to select the proper byte(s) of the memory/I/O word to be read or written according to the following table:

\overline{BHE}	A_0	Characteristics
0	0	Whole word
0	1	Upper byte from/to odd address
1	0	Lower byte from/to even address
1	1	None

I/O ports are addressed in the same manner as memory location. Even addressed bytes are transferred on the D_7-D_0 bus lines and odd addressed bytes on $D_{15}-D_8$.

The basic difference between the interrupt acknowledge cycle and a read cycle is that the interrupt acknowledge signal (\overline{INTA}) is asserted in place of the read (\overline{RD}) signal and the address bus is floated. (See Figure 6.) In the second of two successive \overline{INTA} cycles, a byte of information is read from bus

lines D₇–D₀ as supplied by the interrupt system logic (i.e., 8259A Priority Interrupt Controller). This byte identifies the source (type) of the interrupt. It is multiplied by four and used as a pointer into an interrupt vector lookup table, as described earlier.

BUS TIMING—MEDIUM SIZE SYSTEMS

For medium size systems the MN/ $\overline{M\bar{X}}$ pin is connected to V_{SS} and the 8288 Bus Controller is added to the system as well as a latch for latching the system address, and a transceiver to allow for bus loading greater than the 8086 is capable of handling. Signals ALE, DEN, and DT/ \overline{R} are generated by the 8288 instead of the processor in this configuration although their timing remains relatively the same. The 8086 status outputs ($\overline{S_2}$, $\overline{S_1}$, and $\overline{S_0}$) provide type-of-cycle information and become 8288 inputs. This bus cycle information specifies read (code, data, or I/O), write (data or I/O), interrupt

acknowledge, or software halt. The 8288 thus issues control signals specifying memory read or write, I/O read or write, or interrupt acknowledge. The 8288 provides two types of write strobes, normal and advanced, to be applied as required. The normal write strobes have data valid at the leading edge of write. The advanced write strobes have the same timing as read strobes, and hence data isn't valid at the leading edge of write. The transceiver receives the usual DIR and \overline{G} inputs from the 8288's DT/ \overline{R} and DEN.

The pointer into the interrupt vector table, which is passed during the second INTA cycle, can derive from an 8259A located on either the local bus or the system bus. If the master 8259A Priority Interrupt Controller is positioned on the local bus, a TTL gate is required to disable the transceiver when reading from the master 8259A during the interrupt acknowledge sequence and software "poll".

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias 0°C to 70°C
 Storage Temperature -65°C to +150°C
 Voltage on Any Pin with
 Respect to Ground -1.0V to +7V
 Power Dissipation 2.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (8086: $T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5\text{V} \pm 10\%$)
 (8086-1: $T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5\text{V} \pm 5\%$)
 (8086-2: $T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5\text{V} \pm 5\%$)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5	+0.8	V	(Note 1)
V_{IH}	Input High Voltage	2.0	$V_{CC} + 0.5$	V	(Notes 1, 2)
V_{OL}	Output Low Voltage		0.45	V	$I_{OL} = 2.5\text{ mA}$
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -400\ \mu\text{A}$
I_{CC}	Power Supply Current: 8086 8086-1 8086-2		340 360 350	mA	$T_A = 25^\circ\text{C}$
I_{LI}	Input Leakage Current		± 10	μA	$0\text{V} \leq V_{IN} \leq V_{CC}$ (Note 3)
I_{LO}	Output Leakage Current		± 10	μA	$0.45\text{V} \leq V_{OUT} \leq V_{CC}$
V_{CL}	Clock Input Low Voltage	-0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.9	$V_{CC} + 1.0$	V	
C_{IN}	Capacitance of Input Buffer (All input except AD_0 - AD_{15} , $\overline{RQ}/\overline{GT}$)		15	pF	$f_c = 1\text{ MHz}$
C_{IO}	Capacitance of I/O Buffer (AD_0 - AD_{15} , $\overline{RQ}/\overline{GT}$)		15	pF	$f_c = 1\text{ MHz}$

NOTES:

- V_{IL} tested with $\overline{MN}/\overline{MX}$ Pin = 0V. V_{IH} tested with $\overline{MN}/\overline{MX}$ Pin = 5V. $\overline{MN}/\overline{MX}$ Pin is a Strap Pin.
- Not applicable to $\overline{RQ}/\overline{GT0}$ and $\overline{RQ}/\overline{GT1}$ (Pins 30 and 31).
- HOLD and HLDA I_{LI} min = 30 μA , max = 500 μA .

A.C. CHARACTERISTICS (8086: $T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5\text{V} \pm 10\%$)
 (8086-1: $T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5\text{V} \pm 5\%$)
 (8086-2: $T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5\text{V} \pm 5\%$)

MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

Symbol	Parameter	8086		8086-1		8086-2		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
TCLCL	CLK Cycle Period	200	500	100	500	125	500	ns	
TCLCH	CLK Low Time	118		53		68		ns	
TCHCL	CLK High Time	69		39		44		ns	
TCH1CH2	CLK Rise Time		10		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		5		20		ns	
TCLDX	Data in Hold Time	10		10		10		ns	
TR1VCL	RDY Setup Time into 8284A (See Notes 1, 2)	35		35		35		ns	
TCLR1X	RDY Hold Time into 8284A (See Notes 1, 2)	0		0		0		ns	
TRYHCH	READY Setup Time into 8086	118		53		68		ns	
TCHRYX	READY Hold Time into 8086	30		20		20		ns	
TRYLCL	READY Inactive to CLK (See Note 3)	-8		-10		-8		ns	
THVCH	HOLD Setup Time	35		20		20		ns	
TINVCH	INTR, NMI, $\overline{\text{TEST}}$ Setup Time (See Note 2)	30		15		15		ns	
TILIH	Input Rise Time (Except CLK)		20		20		20	ns	
TIHIL	Input Fall Time (Except CLK)		12		12		12	ns	From 2.0V to 0.8V

A.C. CHARACTERISTICS (Continued)

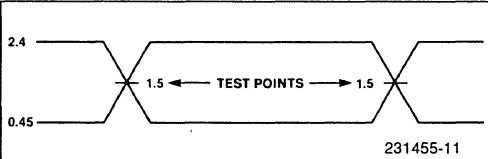
TIMING RESPONSES

Symbol	Parameter	8086		8086-1		8086-2		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
TCLAV	Address Valid Delay	10	110	10	50	10	60	ns	*C _L = 20–100 pF for all 8086 Outputs (In addition to 8086 selfload)
TCLAX	Address Hold Time	10		10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	10	40	TCLAX	50	ns	
TLHLL	ALE Width ¹	TCLCH-20		TCLCH-10		TCLCH-10		ns	
TCLLH	ALE Active Delay		80		40		50	ns	
TCHLL	ALE Inactive Delay		85		45		55	ns	
TLLAX	Address Hold Time	TCHCL-10		TCHCL-10		TCHCL-10		ns	
TCLDV	Data Valid Delay	10	110	10	50	10	60	ns	
TCHDX	Data Hold Time	10		10		10		ns	
TWHDX	Data Hold Time After WR	TCLCH-30		TCLCH-25		TCLCH-30		ns	
TCVCTV	Control Active Delay 1	10	110	10	50	10	70	ns	
TCHCTV	Control Active Delay 2	10	110	10	45	10	60	ns	
TCVCTX	Control Inactive Delay	10	110	10	50	10	70	ns	
TAZRL	Address Float to READ Active	0		0		0		ns	
TCLRL	\overline{RD} Active Delay	10	165	10	70	10	100	ns	
TCLRH	\overline{RD} Inactive Delay	10	150	10	60	10	80	ns	
TRHAV	\overline{RD} Inactive to Next Address Active	TCLCL-45		TCLCL-35		TCLCL-40		ns	
TCLHAV	HLDA Valid Delay	10	160	10	60	10	100	ns	
TRLRH	\overline{RD} Width	2TCLCL-75		2TCLCL-40		2TCLCL-50		ns	
TWLWH	\overline{WR} Width	2TCLCL-60		2TCLCL-35		2TCLCL-40		ns	
TAVAL	Address Valid to ALE Low	TCLCH-60		TCLCH-35		TCLCH-40		ns	
TOLOH	Output Rise Time		20		20		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12		12	ns	From 2.0V to 0.8V

NOTES:

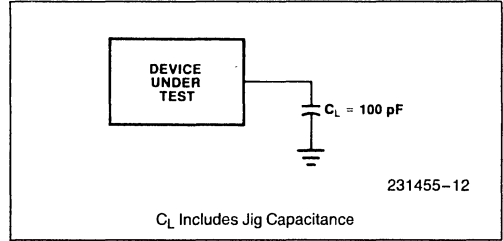
1. Signal at 8284A shown for reference only.
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T2 state. (8 ns into T3).

A.C. TESTING INPUT, OUTPUT WAVEFORM



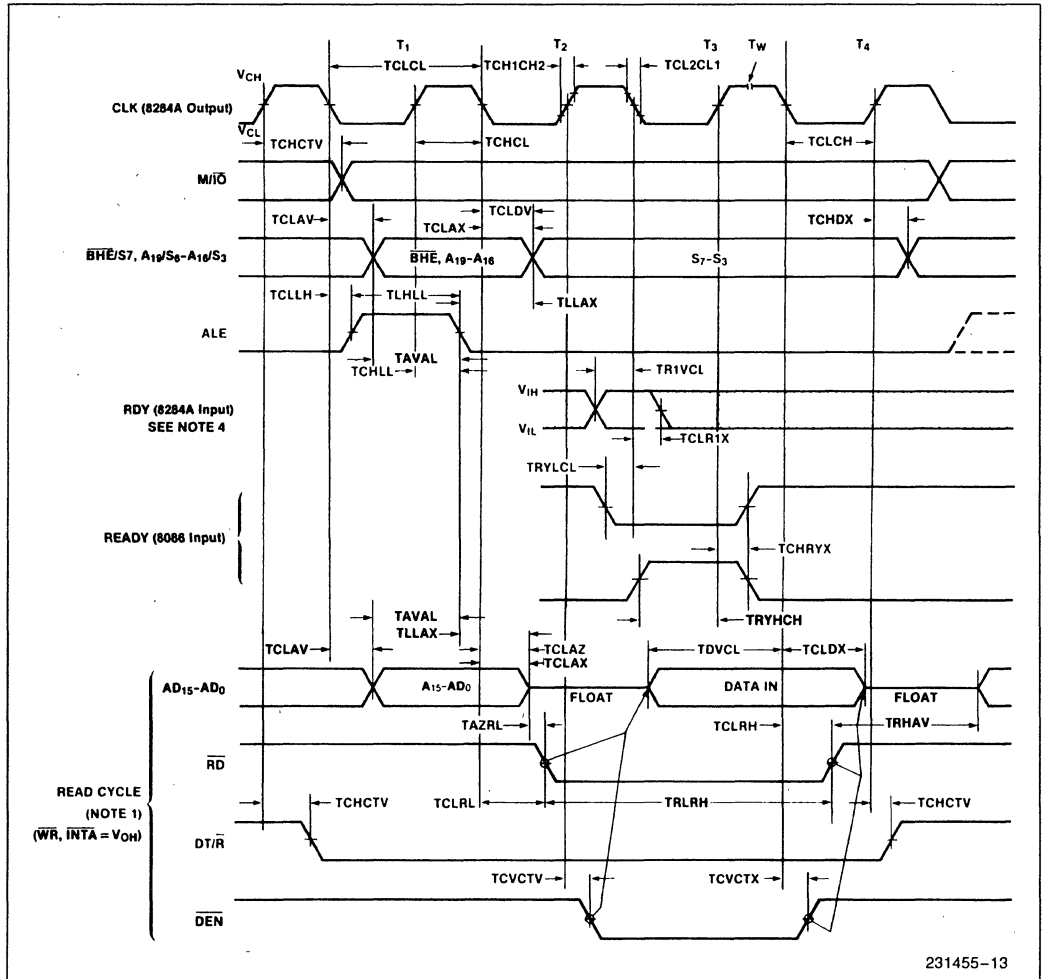
A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 1.5V for both a Logic "1" and "0".

A.C. TESTING LOAD CIRCUIT



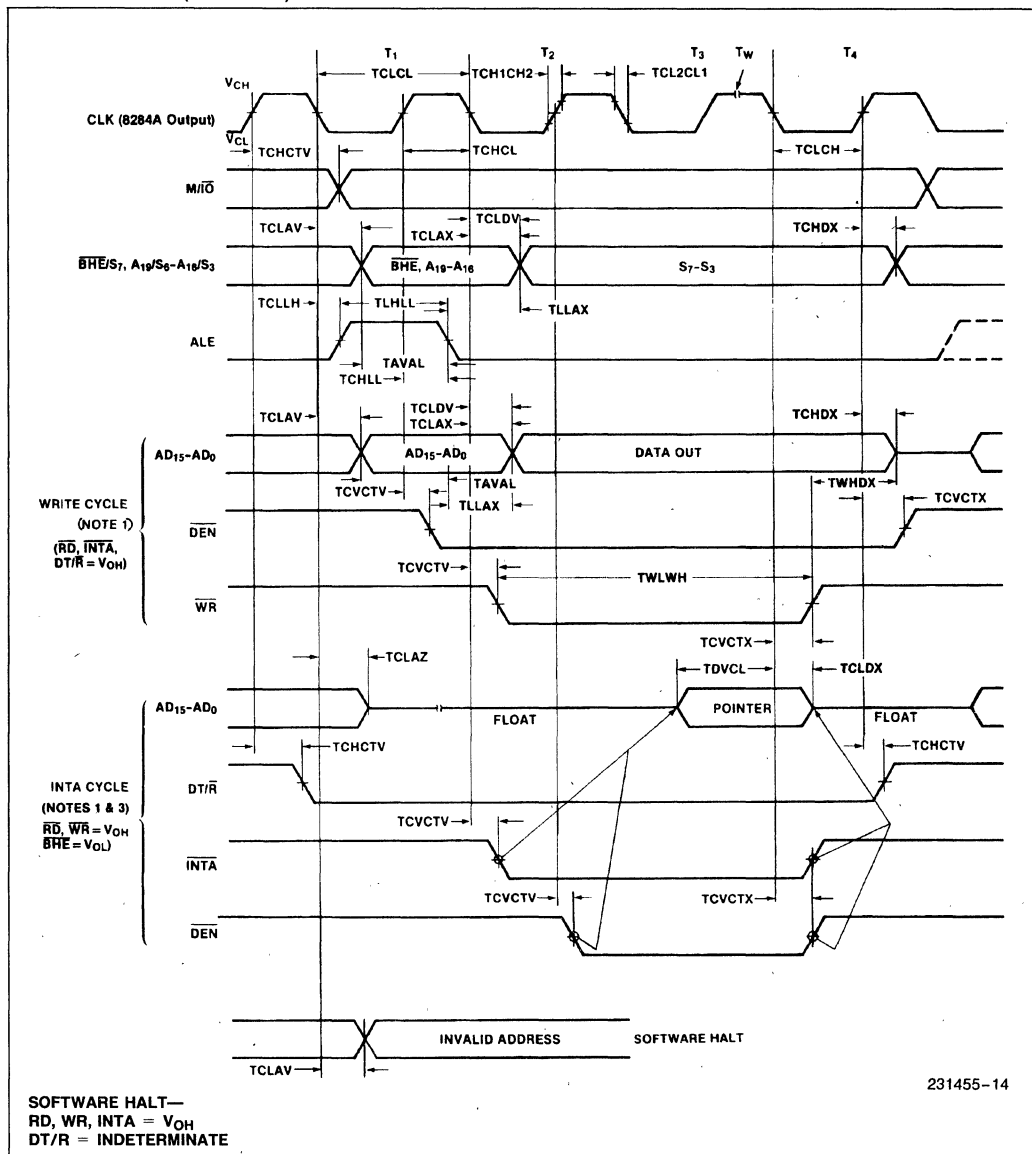
WAVEFORMS

MINIMUM MODE



WAVEFORMS (Continued)

MINIMUM MODE (Continued)



NOTES:

1. All signals switch between V $_{OH}$ and V $_{OL}$ unless otherwise specified.
2. RDY is sampled near the end of T $_2$, T $_3$, T $_w$ to determine if T $_w$ machines states are to be inserted.
3. Two INTA cycles run back-to-back. The 8086 LOCAL ADDR/DATA BUS is floating during both INTA cycles. Control signals shown for second INTA cycle.
4. Signals at 8284A are shown for reference only.
5. All timing measurements are made at 1.5V unless otherwise noted.

A.C. CHARACTERISTICS
**MAX MODE SYSTEM (USING 8288 BUS CONTROLLER)
TIMING REQUIREMENTS**

Symbol	Parameter	8086		8086-1		8086-2		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
TCLCL	CLK Cycle Period	200	500	100	500	125	500	ns	
TCLCH	CLK Low Time	118		53		68		ns	
TCHCL	CLK High Time	69		39		44		ns	
TCH1CH2	CLK Rise Time		10		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		5		20		ns	
TCLDX	Data in Hold Time	10		10		10		ns	
TR1VCL	RDY Setup Time into 8284A (Notes 1, 2)	35		35		35		ns	
TCLR1X	RDY Hold Time into 8284A (Notes 1, 2)	0		0		0		ns	
TRYHCH	READY Setup Time into 8086	118		53		68		ns	
TCHRYX	READY Hold Time into 8086	30		20		20		ns	
TRYLCL	READY Inactive to CLK (Note 4)	-8		-10		-8		ns	
TINVCH	Setup Time for Recognition (INTR, NMI, TEST) (Note 2)	30		15		15		ns	
TGVCH	$\overline{RQ}/\overline{GT}$ Setup Time (Note 5)	30		15		15		ns	
TCHGX	\overline{RQ} Hold Time into 8086	40		20		30		ns	
TILIH	Input Rise Time (Except CLK)		20		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12		12	ns	From 2.0V to 0.8V

A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

Symbol	Parameter	8086		8086-1		8086-2		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
TCLML	Command Active Delay (See Note 1)	10	35	10	35	10	35	ns	C _L = 20–100 pF for all 8086 Outputs (in addition to 8086 self-load)
TCLMH	Command Inactive Delay (See Note 1)	10	35	10	35	10	35	ns	
TRYHSH	READY Active to Status Passive (See Note 3)		110		45		65	ns	
TCHSV	Status Active Delay	10	110	10	45	10	60	ns	
TCLSH	Status Inactive Delay	10	130	10	55	10	70	ns	
TCLAV	Address Valid Delay	10	110	10	50	10	60	ns	
TCLAX	Address Hold Time	10		10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	10	40	TCLAX	50	ns	
TSVLH	Status Valid to ALE High (See Note 1)		15		15		15	ns	
TSMVCH	Status Valid to MCE High (See Note 1)		15		15		15	ns	
TCLLH	CLK Low to ALE Valid (See Note 1)		15		15		15	ns	
TCLMCH	CLK Low to MCE High (See Note 1)		15		15		15	ns	
TCHLL	ALE Inactive Delay (See Note 1)		15		15		15	ns	
TCLMCL	MCE Inactive Delay (See Note 1)		15		15		15	ns	
TCLDV	Data Valid Delay	10	110	10	50	10	60	ns	
TCHDX	Data Hold Time	10		10		10		ns	
TCVNV	Control Active Delay (See Note 1)	5	45	5	45	5	45	ns	
TCVNX	Control Inactive Delay (See Note 1)	10	45	10	45	10	45	ns	
TAZRL	Address Float to READ Active	0		0		0		ns	
TCLRL	RD Active Delay	10	165	10	70	10	100	ns	
TCLRH	RD Inactive Delay	10	150	10	60	10	80	ns	

A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES (Continued)

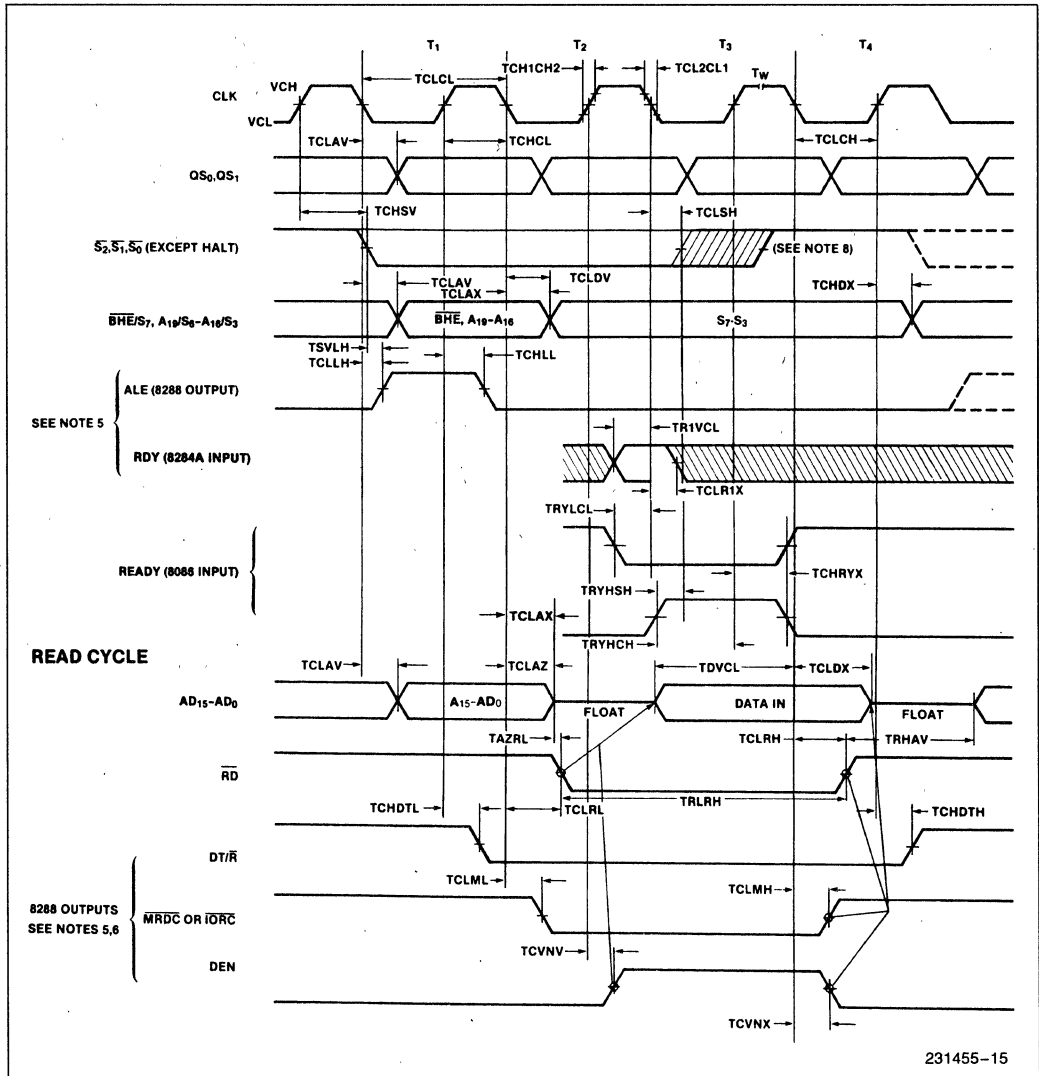
Symbol	Parameter	8086		8086-1		8086-2		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
TRHAV	RD Inactive to Next Address Active	TCLCL-45		TCLCL-35		TCLCL-40		ns	C _L = 20–100 pF for all 8086 Outputs (In addition to 8086 self-load)
TCHDTL	Direction Control Active Delay (Note 1)		50		50		50	ns	
TCHDTH	Direction Control Inactive Delay (Note 1)		30		30		30	ns	
TCLGL	GT Active Delay (Note 5)	0	85	0	38	0	50	ns	
TCLGH	GT Inactive Delay	0	85	0	45	0	50	ns	
TRLRH	RD Width	2TCLCL-75		2TCLCL-40		2TCLCL-50		ns	
TOLOH	Output Rise Time		20		20		20	ns	
TOHOL	Output Fall Time		12		12		12	ns	From 2.0V to 0.8V

NOTES:

1. Signal at 8284A or 8288 shown for reference only.
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T3 and wait states.
4. Applies only to T2 state (8 ns into T3).
5. Change from 1985 Handbook.

WAVEFORMS

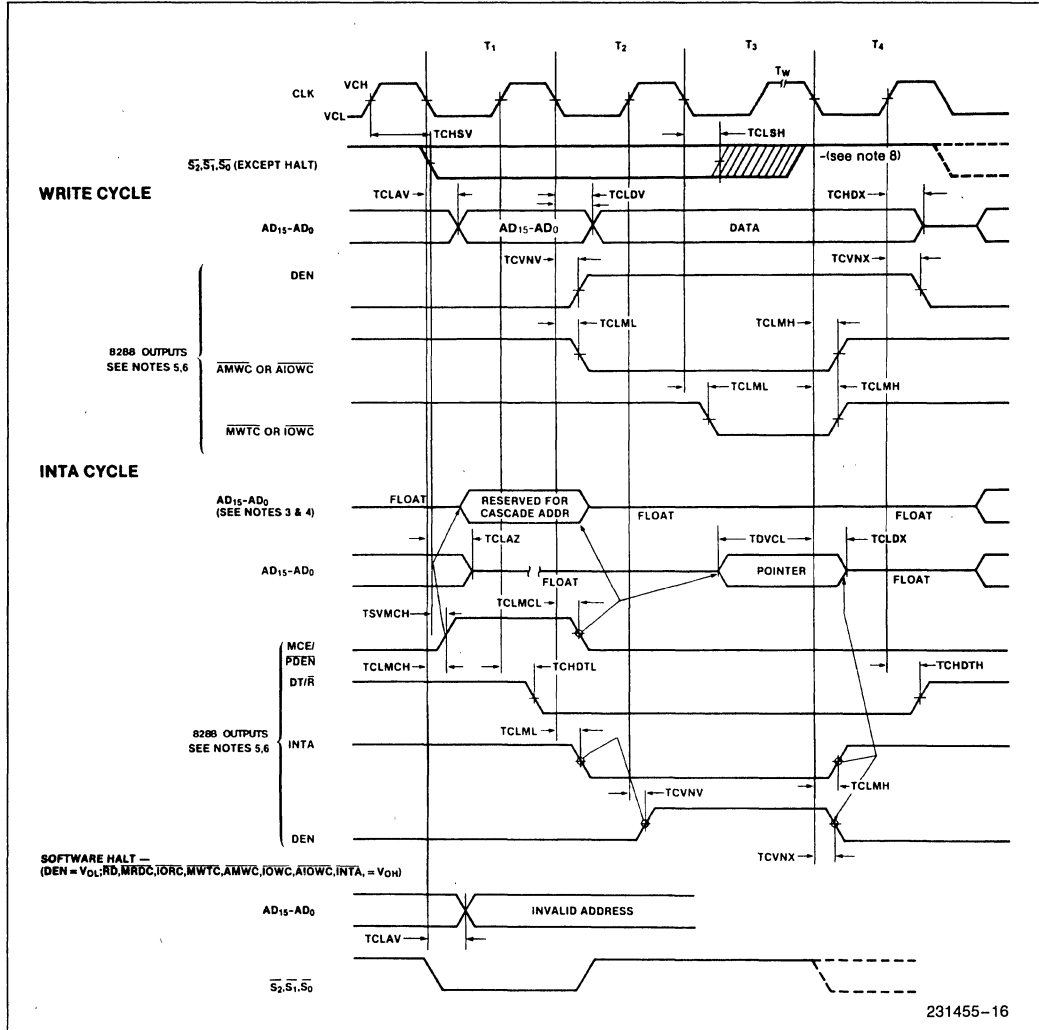
MAXIMUM MODE



231455-15

WAVEFORMS (Continued)

MAXIMUM MODE (Continued)



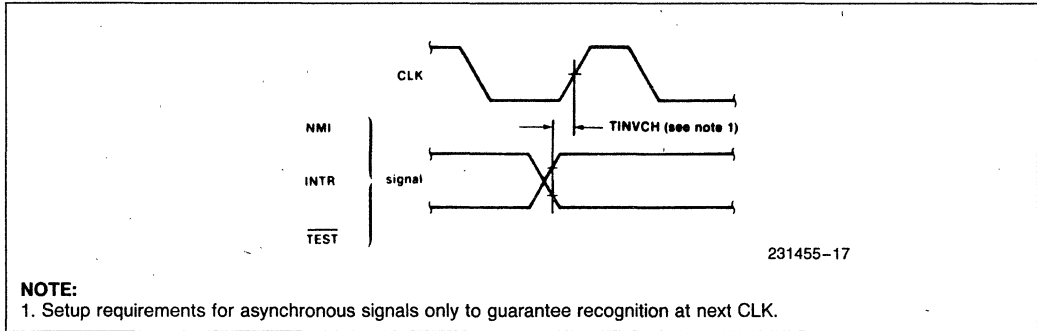
231455-16

NOTES:

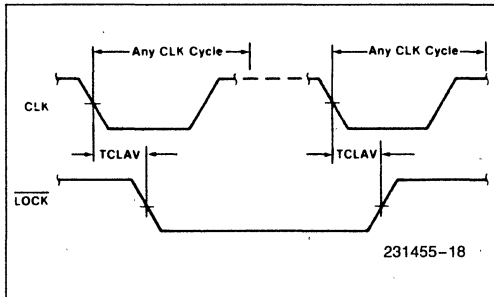
1. All signals switch between V_{OH} and V_{OL} unless otherwise specified.
2. RDY is sampled near the end of T_2 , T_3 , T_w to determine if T_W machines states are to be inserted.
3. Cascade address is valid between first and second INTA cycle.
4. Two INTA cycles run back-to-back. The 8086 LOCAL ADDR/DATA BUS is floating during both INTA cycles. Control for pointer address is shown for second INTA cycle.
5. Signals at 8284A or 8288 are shown for reference only.
6. The issuance of the 8288 command and control signals (\overline{MRDC} , \overline{MWTC} , \overline{AMWC} , \overline{IORC} , \overline{IOWC} , \overline{AIOWC} , \overline{INTA} and DEN) lags the active high 8288 CEN.
7. All timing measurements are made at 1.5V unless otherwise noted.
8. Status inactive in state just prior to T_4 .

WAVEFORMS (Continued)

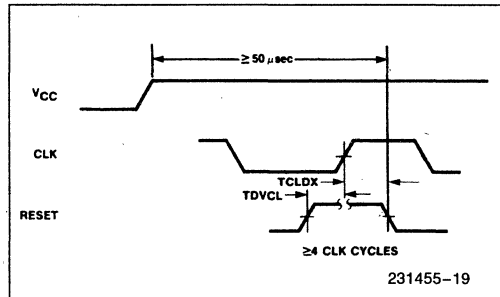
ASYNCHRONOUS SIGNAL RECOGNITION



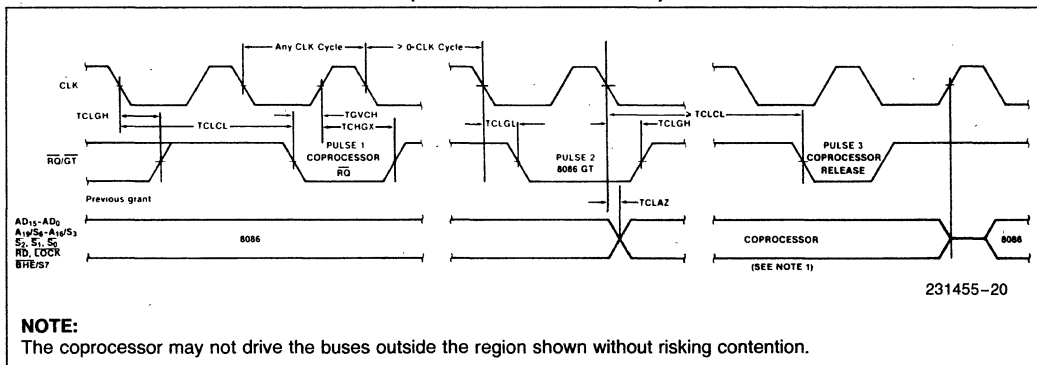
BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)



RESET TIMING



REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)



WAVEFORMS (Continued)

HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)

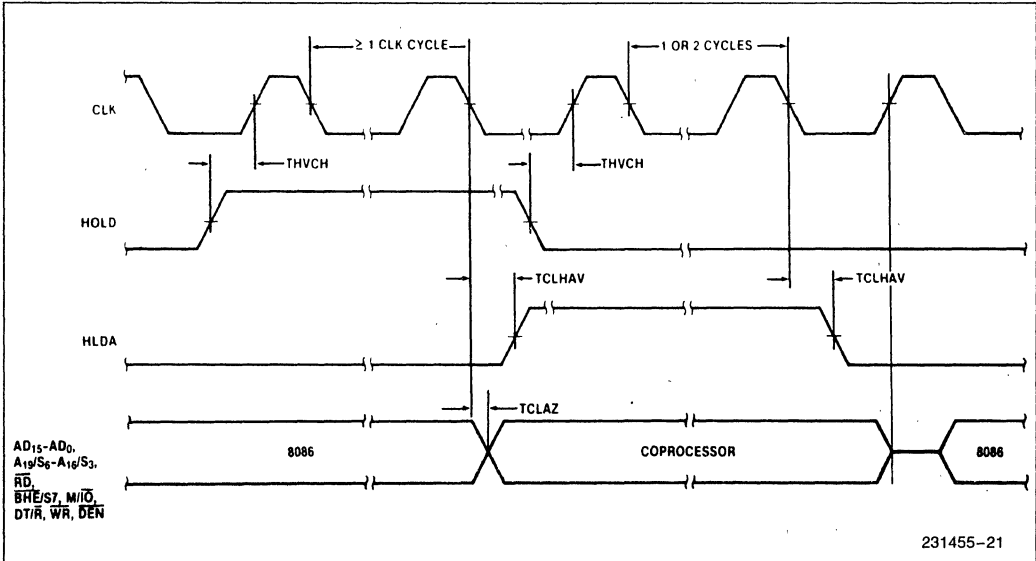


Table 2. Instruction Set Summary

Mnemonic and Description	Instruction Code			
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
DATA TRANSFER				
MOV = Move:				
Register/Memory to/from Register	1 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate to Register	1 0 1 1 w reg	data	data if w = 1	
Memory to Accumulator	1 0 1 0 0 0 w	addr-low	addr-high	
Accumulator to Memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/Memory to Segment Register	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment Register to Register/Memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		
PUSH = Push:				
Register/Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m		
Register	0 1 0 1 0 reg			
Segment Register	0 0 0 reg 1 1 0			
POP = Pop:				
Register/Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m		
Register	0 1 0 1 1 reg			
Segment Register	0 0 0 reg 1 1 1			
XCHG = Exchange:				
Register/Memory with Register	1 0 0 0 0 1 1 w	mod reg r/m		
Register with Accumulator	1 0 0 1 0 reg			
IN = Input from:				
Fixed Port	1 1 1 0 0 1 0 w	port		
Variable Port	1 1 1 0 1 1 0 w			
OUT = Output to:				
Fixed Port	1 1 1 0 0 1 1 w	port		
Variable Port	1 1 1 0 1 1 1 w			
XLAT = Translate Byte to AL	1 1 0 1 0 1 1 1			
LEA = Load EA to Register	1 0 0 0 1 1 0 1	mod reg r/m		
LDS = Load Pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m		
LES = Load Pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m		
LAHF = Load AH with Flags	1 0 0 1 1 1 1 1			
SAHF = Store AH into Flags	1 0 0 1 1 1 1 0			
PUSHF = Push Flags	1 0 0 1 1 1 0 0			
POPF = Pop Flags	1 0 0 1 1 1 0 1			

Table 2. Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code			
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
ARITHMETIC				
ADD = Add:				
Reg./Memory with Register to Either	0 0 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s: w = 01
Immediate to Accumulator	0 0 0 0 0 1 0 w	data	data if w = 1	
ADC = Add with Carry:				
Reg./Memory with Register to Either	0 0 0 1 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s: w = 01
Immediate to Accumulator	0 0 0 1 0 1 0 w	data	data if w = 1	
INC = Increment:				
Register/Memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m		
Register	0 1 0 0 0 reg			
AAA = ASCII Adjust for Add	0 0 1 1 0 1 1 1			
BAA = Decimal Adjust for Add	0 0 1 0 0 1 1 1			
SUB = Subtract:				
Reg./Memory and Register to Either	0 0 1 0 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s: w = 01
Immediate from Accumulator	0 0 1 0 1 1 0 w	data	data if w = 1	
SSB = Subtract with Borrow				
Reg./Memory and Register to Either	0 0 0 1 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s: w = 01
Immediate from Accumulator	0 0 0 1 1 1 w	data	data if w = 1	
DEC = Decrement:				
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m		
Register	0 1 0 0 1 reg			
NEG = Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		
CMP = Compare:				
Register/Memory and Register	0 0 1 1 1 0 d w	mod reg r/m		
Immediate with Register/Memory	1 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s: w = 01
Immediate with Accumulator	0 0 1 1 1 1 0 w	data	data if w = 1	
AAS = ASCII Adjust for Subtract	0 0 1 1 1 1 1 1			
DAS = Decimal Adjust for Subtract	0 0 1 0 1 1 1 1			
MUL = Multiply (Unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m		
IMUL = Integer Multiply (Signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m		
AAM = ASCII Adjust for Multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		
DIV = Divide (Unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m		
IDIV = Integer Divide (Signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m		
AAD = ASCII Adjust for Divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		
CBW = Convert Byte to Word	1 0 0 1 1 0 0 0			
CWD = Convert Word to Double Word	1 0 0 1 1 0 0 1			

Table 2. Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code			
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
LOGIC				
NOT = Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m		
SHL/SAL = Shift Logical/Arithmetic Left	1 1 0 1 0 0 v w	mod 1 0 0 r/m		
SHR = Shift Logical Right	1 1 0 1 0 0 v w	mod 1 0 1 r/m		
SAR = Shift Arithmetic Right	1 1 0 1 0 0 v w	mod 1 1 1 r/m		
ROL = Rotate Left	1 1 0 1 0 0 v w	mod 0 0 0 r/m		
ROR = Rotate Right	1 1 0 1 0 0 v w	mod 0 0 1 r/m		
RCL = Rotate Through Carry Flag Left	1 1 0 1 0 0 v w	mod 0 1 0 r/m		
RCR = Rotate Through Carry Right	1 1 0 1 0 0 v w	mod 0 1 1 r/m		
AND = And:				
Reg./Memory and Register to Either	0 0 1 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w = 1
Immediate to Accumulator	0 0 1 0 0 1 0 w		data	data if w = 1
TEST = And Function to Flags, No Result:				
Register/Memory and Register	1 0 0 0 1 0 w	mod reg r/m		
Immediate Data and Register/Memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate Data and Accumulator	1 0 1 0 1 0 0 w		data	data if w = 1
OR = Or:				
Reg./Memory and Register to Either	0 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1
Immediate to Accumulator	0 0 0 0 1 1 0 w		data	data if w = 1
XOR = Exclusive or:				
Reg./Memory and Register to Either	0 0 1 1 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1
Immediate to Accumulator	0 0 1 1 0 1 0 w		data	data if w = 1
STRING MANIPULATION				
REP = Repeat	1 1 1 1 0 0 1 z			
MOVS = Move Byte/Word	1 0 1 0 0 1 0 w			
CMPS = Compare Byte/Word	1 0 1 0 0 1 1 w			
SCAS = Scan Byte/Word	1 0 1 0 1 1 1 w			
LODS = Load Byte/Wd to AL/AX	1 0 1 0 1 1 0 w			
STOS = Stor Byte/Wd from AL/A	1 0 1 0 1 0 1 w			
CONTROL TRANSFER				
CALL = Call:				
Direct within Segment	1 1 1 0 1 0 0 0	disp-low	disp-high	
Indirect within Segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m		
Direct Intersegment	1 0 0 1 1 0 1 0	offset-low	offset-high	
		seg-low	seg-high	
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m		

Table 2. Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code		
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
JMP = Unconditional Jump:			
Direct within Segment	1 1 1 0 1 0 0 1	disp-low	disp-high
Direct within Segment-Short	1 1 1 0 1 0 1 1	disp	
Indirect within Segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	
Direct Intersegment	1 1 1 0 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	
RET = Return from CALL:			
Within Segment	1 1 0 0 0 0 1 1		
Within Seg Adding Immed to SP	1 1 0 0 0 0 1 0	data-low	data-high
Intersegment	1 1 0 0 1 0 1 1		
Intersegment Adding Immediate to SP	1 1 0 0 1 0 1 0	data-low	data-high
JE/JZ = Jump on Equal/Zero	0 1 1 1 0 1 0 0	disp	
JL/JNGE = Jump on Less/Not Greater or Equal	0 1 1 1 1 1 0 0	disp	
JLE/JNG = Jump on Less or Equal/Not Greater	0 1 1 1 1 1 1 0	disp	
JB/JNAE = Jump on Below/Not Above or Equal	0 1 1 1 0 0 1 0	disp	
JBE/JNA = Jump on Below or Equal/Not Above	0 1 1 1 0 1 1 0	disp	
JP/JPE = Jump on Parity/Parity Even	0 1 1 1 1 0 1 0	disp	
JO = Jump on Overflow	0 1 1 1 0 0 0 0	disp	
JS = Jump on Sign	0 1 1 1 1 0 0 0	disp	
JNE/JNZ = Jump on Not Equal/Not Zero	0 1 1 1 0 1 0 1	disp	
JNL/JGE = Jump on Not Less/Greater or Equal	0 1 1 1 1 1 0 1	disp	
JNLE/JG = Jump on Not Less or Equal/Greater	0 1 1 1 1 1 1 1	disp	
JNB/JAE = Jump on Not Below/Above or Equal	0 1 1 1 0 0 1 1	disp	
JNBE/JA = Jump on Not Below or Equal/Above	0 1 1 1 0 1 1 1	disp	
JNP/JPO = Jump on Not Par/Par Odd	0 1 1 1 1 0 1 1	disp	
JNO = Jump on Not Overflow	0 1 1 1 0 0 0 1	disp	
JNS = Jump on Not Sign	0 1 1 1 1 0 0 1	disp	
LOOP = Loop CX Times	1 1 1 0 0 0 1 0	disp	
LOOPZ/LOOPE = Loop While Zero/Equal	1 1 1 0 0 0 0 1	disp	
LOOPNZ/LOOPNE = Loop While Not Zero/Equal	1 1 1 0 0 0 0 0	disp	
JCZ = Jump on CX Zero	1 1 1 0 0 0 1 1	disp	
INT = Interrupt			
Type Specified	1 1 0 0 1 1 0 1	type	
Type 3	1 1 0 0 1 1 0 0		
INTO = Interrupt on Overflow	1 1 0 0 1 1 1 0		
IRET = Interrupt Return	1 1 0 0 1 1 1 1		

Table 2. Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code	
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
PROCESSOR CONTROL		
CLC = Clear Carry	1 1 1 1 1 0 0 0	
CMC = Complement Carry	1 1 1 1 0 1 0 1	
STC = Set Carry	1 1 1 1 1 0 0 1	
CLD = Clear Direction	1 1 1 1 1 1 0 0	
STD = Set Direction	1 1 1 1 1 1 0 1	
CLI = Clear Interrupt	1 1 1 1 1 0 1 0	
STI = Set Interrupt	1 1 1 1 1 0 1 1	
HLT = Halt	1 1 1 1 0 1 0 0	
WAIT = Wait	1 0 0 1 1 0 1 1	
ESC = Escape (to External Device)	1 1 0 1 1 x x x	mod x x x r/m
LOCK = Bus Lock Prefix	1 1 1 1 0 0 0 0	

NOTES:

AL = 8-bit accumulator
 AX = 16-bit accumulator
 CX = Count register
 DS = Data segment
 ES = Extra segment
 Above/below refers to unsigned value
 Greater = more positive;
 Less = less positive (more negative) signed values
 if d = 1 then "to" reg; if d = 0 then "from" reg
 if w = 1 then word instruction; if w = 0 then byte instruction
 if mod = 11 then r/m is treated as a REG field
 if mod = 00 then DISP = 0*, disp-low and disp-high are absent
 if mod = 01 then DISP = disp-low sign-extended to 16 bits, disp-high is absent
 if mod = 10 then DISP = disp-high; disp-low
 if r/m = 000 then EA = (BX) + (SI) + DISP
 if r/m = 001 then EA = (BX) + (DI) + DISP
 if r/m = 010 then EA = (BP) + (SI) + DISP
 if r/m = 011 then EA = (BP) + (DI) + DISP
 if r/m = 100 then EA = (SI) + DISP
 if r/m = 101 then EA = (DI) + DISP
 if r/m = 110 then EA = (BP) + DISP*
 if r/m = 111 then EA = (BX) + DISP
 DISP follows 2nd byte of instruction (before data if required)
 *except if mod = 00 and r/m = 110 then EA = disp-high; disp-low.

if s w = 01 then 16 bits of immediate data form the operand
 if s w = 11 then an immediate data byte is sign extended to form the 16-bit operand
 if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
 x = don't care
 z is used for string primitives for comparison with ZF FLAG

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:
 FLAGS = X:X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

Mnemonics © Intel, 1978

DATA SHEET REVISION REVIEW

The following list represents key differences between this and the -003 data sheet. Please review this summary carefully.

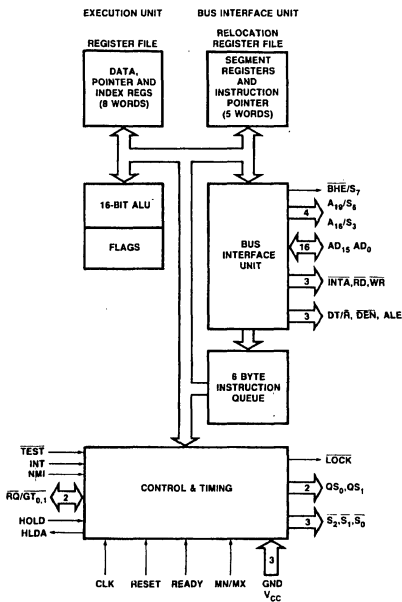
1. The Pin Description Table has been modified to indicate that the HOLD and HLDA pins both have internal pull-up resistors. The input leakage current (I_{LI}) in the D.C. CHARACTERISTICS section has been modified for these pins.



80C86A 16-BIT CHMOS MICROPROCESSOR

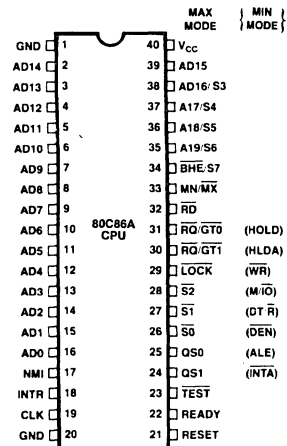
- Pin-for-Pin and Functionally Compatible to Industry Standard HMOS 8086
- Fully Static Design with Frequency Range from D.C. to:
 - 8 MHz for 80C86A-2
- Low Power Operation
 - Operating $I_{CC} = 10 \text{ mA/MHz}$
 - Standby $I_{CCS} = 500 \mu\text{A max}$
- Bus-Hold Circuitry Eliminates Pull-Up Resistors
- Direct Addressing Capability of 1 MByte of Memory
- Architecture Designed for Powerful Assembly Language and Efficient High Level Languages
- 24 Operand Addressing Modes
- Byte, Word and Block Operations
- 8 and 16-Bit Signed and Unsigned Arithmetic
 - Binary or Decimal
 - Multiply and Divide
- Available in 40-Lead Plastic DIP

The Intel 80C86A is a high performance, CHMOS version of the industry standard HMOS 8086 16-bit CPU. The 80C86A available in 8 MHz clock rates, offers two modes of operation: MINimum for small systems and MAXimum for larger applications such as multiprocessing. It is available in 40-pin DIP package.



**Figure 1. 80C86A
CPU Block Diagram**

240029-1



**Figure 2. 80C86A
40-Lead DIP Configuration**

240029-2

Table 1. Pin Description

The following pin function descriptions are for 80C86AA systems in either minimum or maximum mode. The "Local Bus" in these descriptions is the direct multiplexed bus interface connection to the 80C86A (without regard to additional bus buffers).

Symbol	Pin No.	Type	Name and Function																		
AD ₁₅ -AD ₀	2-16, 39	I/O	<p>ADDRESS DATA BUS: These lines constitute the time multiplexed memory/I/O address (T₁) and data (T₂, T₃, T_W, T₄) bus. A₀ is analogous to $\overline{\text{BHE}}$ for the lower byte of the data bus, pins D₇-D₀. It is LOW during T₁ when a byte is to be transferred on the lower portion of the bus in memory or I/O operations. Eight-bit oriented devices tied to the lower half would normally use A₀ to condition chip select functions. (See $\overline{\text{BHE}}$.) These lines are active HIGH and float to 3-state OFF⁽¹⁾ during interrupt acknowledge and local bus "hold acknowledge."</p>																		
A ₁₉ /S ₆ , A ₁₈ /S ₅ , A ₁₇ /S ₄ , A ₁₆ /S ₃	35-38	O	<p>ADDRESS/STATUS: During T₁ these are the four most significant address lines for memory operations. During I/O operations these lines are LOW. During memory and I/O operations, status information is available on these lines during T₂, T₃, T_W, and T₄. The status of the interrupt enable FLAG bit (S₅) is updated at the beginning of each CLK cycle. A₁₇/S₄ and A₁₆/S₃ are encoded as shown.</p> <p>This information indicates which relocation register is presently being used for data accessing.</p> <p>These lines float to 3-state OFF⁽¹⁾ during local bus "hold acknowledge."</p> <table border="1" data-bbox="495 928 1146 1137"> <thead> <tr> <th>A₁₇/S₄</th> <th>A₁₆/S₃</th> <th>Characteristics</th> </tr> </thead> <tbody> <tr> <td>0 (LOW)</td> <td>0</td> <td>Alternate Data</td> </tr> <tr> <td>0</td> <td>1</td> <td>Stack</td> </tr> <tr> <td>1 (HIGH)</td> <td>0</td> <td>Code or None</td> </tr> <tr> <td>1</td> <td>1</td> <td>Data</td> </tr> <tr> <td colspan="3">S₆ is 0 (LOW)</td> </tr> </tbody> </table>	A ₁₇ /S ₄	A ₁₆ /S ₃	Characteristics	0 (LOW)	0	Alternate Data	0	1	Stack	1 (HIGH)	0	Code or None	1	1	Data	S ₆ is 0 (LOW)		
A ₁₇ /S ₄	A ₁₆ /S ₃	Characteristics																			
0 (LOW)	0	Alternate Data																			
0	1	Stack																			
1 (HIGH)	0	Code or None																			
1	1	Data																			
S ₆ is 0 (LOW)																					
BHE/S ₇	34	O	<p>BUS HIGH ENABLE/STATUS: During T₁ the bus high enable signal (BHE) should be used to enable data onto the most significant half of the data bus, pins D₁₅-D₈. Eight-bit oriented devices tied to the upper half of the bus would normally use $\overline{\text{BHE}}$ to condition chip select functions. BHE is LOW during T₁ for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. The S₇ status information is available during T₂, T₃, and T₄. The signal is active LOW, and floats to 3-state OFF⁽¹⁾ in "hold." It is LOW during T₁ for the first interrupt acknowledge cycle.</p> <table border="1" data-bbox="495 1397 1146 1604"> <thead> <tr> <th>BHE</th> <th>A₀</th> <th>Characteristics</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Whole word</td> </tr> <tr> <td>0</td> <td>1</td> <td>Upper byte from/ to odd address</td> </tr> <tr> <td>1</td> <td>0</td> <td>Lower byte from/ to even address</td> </tr> <tr> <td>1</td> <td>1</td> <td>None</td> </tr> </tbody> </table>	BHE	A ₀	Characteristics	0	0	Whole word	0	1	Upper byte from/ to odd address	1	0	Lower byte from/ to even address	1	1	None			
BHE	A ₀	Characteristics																			
0	0	Whole word																			
0	1	Upper byte from/ to odd address																			
1	0	Lower byte from/ to even address																			
1	1	None																			

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
\overline{RD}	32	O	READ: Read strobe indicates that the processor is performing a memory of I/O read cycle, depending on the state of the S_2 pin. This signal is used to read devices which reside on the 80C86A local bus. \overline{RD} is active LOW during T_2 , T_3 and T_W of any read cycle, and is guaranteed to remain HIGH in T_2 until the 80C86A local bus has floated. This floats to 3-state OFF in "hold acknowledge."
READY	22	I	READY: is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory/IO is synchronized by the 82C84A Clock Generator to form READY. This signal is active HIGH. The 80C86A READY input is not synchronized. Correct operation is not guaranteed if the setup and hold times are not met.
INTR	18	I	INTERRUPT REQUEST: is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.
\overline{TEST}	23	I	TEST: input is examined by the "Wait" instruction. If the \overline{TEST} input is LOW execution continues, otherwise the processor waits in an "Idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.
NMI	17	I	NON-MASKABLE INTERRUPT: an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from a LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.
RESET	21	I	RESET: causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the Instruction Set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	CLOCK: provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}	40		V_{CC}: + 5V power supply pin.
GND	1, 20		GROUND: Both must be connected.
MN/\overline{MX}	33	I	MINIMUM/MAXIMUM: indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

Table 1. Pin Description (Continued)

The following pin function descriptions are for the 80C86A/82C88 system in maximum mode (i.e., $\overline{MN}/\overline{MX} = V_{SS}$). Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

Symbol	Pin No.	Type	Name and Function			
$\overline{S}_2, \overline{S}_1, \overline{S}_0$	26-28	O	<p>STATUS: active during $T_4, T_1,$ and T_2 and is returned to the passive state (1,1,1) during T_3 or during T_W when READY is HIGH. This status is used by the 82C88 Bus Controller to generate all memory and I/O access control signals. Any change by $\overline{S}_2, \overline{S}_1, \overline{S}_0$ during T_4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle. These signals float to 3-state OFF⁽¹⁾ in "hold acknowledge." These status lines are encoded as shown.</p>			
			\overline{S}_2	\overline{S}_1	\overline{S}_0	Characteristics
			0 (LOW)	0	0	Interrupt Acknowledge
			0	0	1	Read I/O Port
			0	1	0	Write I/O Port
			0	1	1	Halt
			1 (HIGH)	0	0	Code Access
			1	0	1	Read Memory
			1	1	0	Write Memory
			1	1	1	Passive
$\overline{RQ}/\overline{GT}_0, \overline{RQ}/\overline{GT}_1$	30, 31	I/O	<p>REQUEST/GRANT: pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT}_0$ having higher priority than $\overline{RQ}/\overline{GT}_1$. $\overline{RQ}/\overline{GT}$ has an internal pull-up resistor so may be left unconnected. The request/grant sequence is as follows (see timing diagram):</p> <ol style="list-style-type: none"> 1. A pulse of 1 CLK wide from another local bus master indicates a local bus request ("hold") to the 80C86A (pulse 1). 2. During a T_4 or T_1 clock cycle, a pulse 1 CLK wide from the 80C86A to the requesting master (pulse 2), indicates that the 80C86A has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge." 3. A pulse 1 CLK wide from the requesting master indicates to the 80C86A (pulse 3) that the "hold" request is about to end and that 80C86A can reclaim the local bus at the next CLK. <p>Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW.</p> <p>If the request is made while the CPU is performing a memory cycle, it will release the local bus during T_4 of the cycle when all the following conditions are met:</p> <ol style="list-style-type: none"> 1. Request occurs on or before T_2. 2. Current cycle is not the low byte of a word (on an odd address). 3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence. 4. A locked instruction is not currently executing. 			

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function															
			<p>If the local bus is idle when the request is made the two possible events will follow:</p> <ol style="list-style-type: none"> 1. Local bus will be released during the next clock. 2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied. 															
$\overline{\text{LOCK}}$	29	O	<p>LOCK: output indicates that other system bus masters are not to gain control of the system bus while $\overline{\text{LOCK}}$ is active LOW. The $\overline{\text{LOCK}}$ signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state OFF(1) in "hold acknowledge."</p>															
QS_1, QS_0	24, 25	O	<p>QUEUE STATUS: The queue status is valid during the CLK cycle after which the queue operation is performed. QS_1 and QS_0 provide status to allow external tracking of the internal 80C86A instruction queue.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>QS_1</th> <th>QS_0</th> <th>Characteristics</th> </tr> </thead> <tbody> <tr> <td>0 (LOW)</td> <td>0</td> <td>No Operation</td> </tr> <tr> <td>0</td> <td>1</td> <td>First Byte of Op Code from Queue</td> </tr> <tr> <td>1 (HIGH)</td> <td>0</td> <td>Empty the Queue</td> </tr> <tr> <td>1</td> <td>1</td> <td>Subsequent Byte from Queue</td> </tr> </tbody> </table>	QS_1	QS_0	Characteristics	0 (LOW)	0	No Operation	0	1	First Byte of Op Code from Queue	1 (HIGH)	0	Empty the Queue	1	1	Subsequent Byte from Queue
QS_1	QS_0	Characteristics																
0 (LOW)	0	No Operation																
0	1	First Byte of Op Code from Queue																
1 (HIGH)	0	Empty the Queue																
1	1	Subsequent Byte from Queue																

The following pin function descriptions are for the 80C86A in minimum mode (i.e., $\text{MN}/\overline{\text{MX}} = V_{CC}$). Only the pin functions which are unique to minimum mode are described; all other pin functions are described above.

$\text{M}/\overline{\text{IO}}$	28	O	<p>STATUS LINE: logically equivalent to S_2 in the maximum mode. It is used to distinguish a memory access from an I/O access. $\text{M}/\overline{\text{IO}}$ becomes valid in the T_4 preceding a bus cycle and remains valid until the final T_4 of the cycle ($\text{M} = \text{HIGH}$, $\text{IO} = \text{LOW}$). $\text{M}/\overline{\text{IO}}$ floats to 3-state OFF(1) in local bus "hold acknowledge."</p>
$\overline{\text{WR}}$	29	O	<p>WRITE: indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the $\text{M}/\overline{\text{IO}}$ signal. $\overline{\text{WR}}$ is active for T_2, T_3 and T_W of any write cycle. It is active LOW, and floats to 3-state OFF(1) in local bus "hold acknowledge."</p>
$\overline{\text{INTA}}$	24	O	<p>$\overline{\text{INTA}}$ is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T_2, T_3 and T_W of each interrupt acknowledge cycle.</p>
ALE	25	O	<p>ADDRESS LATCH ENABLE: provided by the processor to latch the address into an address latch. It is a HIGH pulse active during T_1 of any bus cycle. Note that ALE is never floated.</p>
$\text{DT}/\overline{\text{R}}$	27	O	<p>DATA TRANSMIT/RECEIVE: needed in minimum system that desires to use a data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically $\text{DT}/\overline{\text{R}}$ is equivalent to $\overline{\text{S}}_1$ in the maximum mode, and its timing is the same as for $\text{M}/\overline{\text{IO}}$. ($\text{T} = \text{HIGH}$, $\text{R} = \text{LOW}$.) This signal floats to 3-state OFF(1) in local bus "hold acknowledge."</p>

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{\text{DEN}}$	26	O	DATA ENABLE: provided as an output enable for the transceiver in a minimum system which uses the transceiver. $\overline{\text{DEN}}$ is active LOW during each memory and I/O access and for INTA cycles. For a read or $\overline{\text{INTA}}$ cycle it is active from the middle of T_2 until the middle of T_4 , while for a write cycle it is active from the beginning of T_2 until the middle of T_4 . $\overline{\text{DEN}}$ floats to 3-state OFF ⁽¹⁾ in local bus "hold acknowledge."
HOLD, HLDA	31, 30	I/O	HOLD: indicates that another master is requesting a local bus "hold." To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement in the middle of a T_4 or T_1 clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor will LOWER the HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. The same rules as for $\overline{\text{RQ}}/\overline{\text{GT}}$ apply regarding when the local bus will be released. HOLD is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the setup time.

NOTE:

1. See the section on Bus Hold Circuitry.

FUNCTIONAL DESCRIPTION

STATIC OPERATION

All 80C86A circuitry is of static design. Internal registers, counters and latches are static and require no refresh as with dynamic circuit design. This eliminates the minimum operating frequency restriction placed on other microprocessors. The CMOS 80C86A can operate from DC to the appropriate upper frequency limit. The processor clock may be stopped in either state (high/low) and held there indefinitely. This type of operation is especially useful for system debug or power critical applications.

The 80C86A can be single stepped using only the CPU clock. This state can be maintained as long as is necessary. Single step clock operation allows simple interface circuitry to provide critical information for bringing up your system.

Static design also allows very low frequency operation. In a power critical situation, this can provide extremely low power operation since 80C86A power dissipation is directly related to operating frequency. As the system frequency is reduced, so is the operating power until, ultimately, at a DC input frequency, the 80C86A power requirement is the standby current.

INTERNAL ARCHITECTURE

The internal functions of the 80C86A processor are partitioned logically into two processing units. The first is the Bus Interface Unit (BIU) and the second is the Execution Unit (EU) as shown in the block diagram of Figure 1.

These units can interact directly but for the most part perform as separate asynchronous operational processors. The bus interface unit provides the functions related to instruction fetching and queuing, operand fetch and store, and address relocation. This unit also provides the basic bus control. The overlap of instruction pre-fetching provided by this unit serves to increase processor performance through improved bus bandwidth utilization. Up to 6 bytes of the instruction stream can be queued while waiting for decoding and execution.

The instruction stream queuing mechanism allows the BIU to keep the memory utilized very efficiently. Whenever there is space for at least 2 bytes in the queue, the BIU will attempt a word fetch memory cycle. This greatly reduces "dead time" on the memory bus. The queue acts as a First-In-First Out (FIFO) buffer, from which the EU extracts instruction bytes as required. If the queue is empty (following a branch instruction, for example), the first byte into the queue immediately becomes available to the EU.

Memory Reference Need	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch.
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
Local Data	DATA (DS)	Data references when: relative to stack, destination of string operation, or explicitly overridden.
External (Global) Data	EXTRA (ES)	Destination of string operations: Explicitly selected using a segment override.

The execution units receives pre-fetched instructions from the BIU queue and provides un-relocated operand addresses to the BIU. Memory operands are passed through the BIU for processing by the EU, which passes results to the BIU for storage. See the Instruction Set description for further register set and architectural descriptions.

MEMORY ORGANIZATION

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64k bytes each, with each segment falling on 16-byte boundaries. (See Figure 3a.)

All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the addressing needs of programs. The segment register to be selected is automatically chosen according to the rules of the following table. All information in one segment type share the same logical attributes (e.g. code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster, and more structured.

Word (16-bit) operands can be located on even or odd address boundaries and are thus not constrained to even boundaries as is the case in many 16-bit computers. For address and data operands, the least significant byte of the word is stored in the lower valued address location and the most significant byte in the next higher address location. The BIU automatically performs the proper number of memory accesses, one if the word operand is on an even byte boundary and two if it is on an odd byte boundary. Except for the performance penalty, this double access is transparent to the software. This performance penalty does not occur for instruction fetches, only word operands.

Physically, the memory is organized as a high bank (D₁₅-D₈) and a low bank (D₇-D₀) of 512k 8-bit bytes addressed in parallel by the processor's address lines.

A₁₉-A₁. Byte data with even addresses is transferred on the D₇-D₀ bus lines while odd addressed byte data (A₀ HIGH) is transferred on the D₁₅-D₈ bus lines. The processor provides two enable signals, \overline{BHE} and A₀, to selectively allow reading from or writing into either an odd byte location, even byte location, or both. The instruction stream is fetched from memory as words and is addressed internally by the processor to the byte level as necessary.

In referencing word data the BIU requires one or two memory cycles depending on whether or not the starting byte of the word is on an even or odd address, respectively. Consequently, in referencing

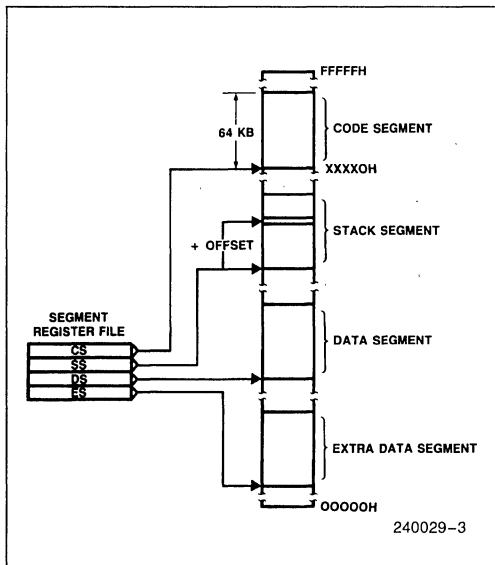


Figure 3a. Memory Organization

word operands performance can be optimized by locating data on even address boundaries. This is an especially useful technique for using the stack, since odd address references to the stack may adversely affect the context switching time for interrupt processing or task multiplexing.

Certain locations in memory are reserved for specific CPU operations (see Figure 3b.) Locations from address FFFF0H through FFFFFH are reserved for operations including a jump to the initial program loading routine. Following RESET, the CPU will always begin execution at location FFFF0H where the jump must be. Locations 00000H through 003FFH are reserved for interrupt operations. Each of the 256 possible interrupt types has its service routine pointed to by a 4-byte pointer element consisting of a 16-bit segment address and a 16-bit offset address. The pointer elements are assumed to have been stored at the respective places in reserved memory prior to occurrence of interrupts.

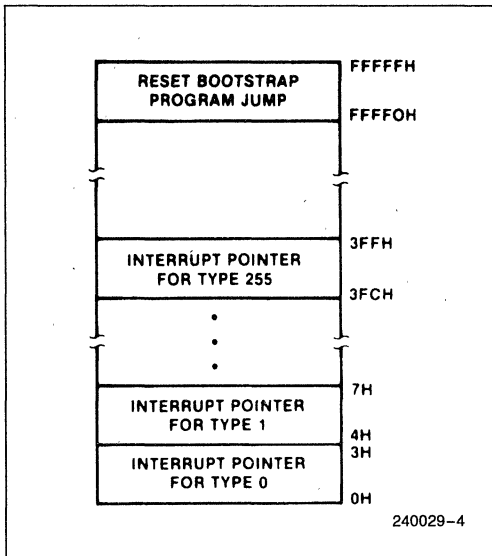


Figure 3b. Reserved Memory Locations

MINIMUM AND MAXIMUM MODES

The requirements for supporting minimum and maximum 80C86A systems are sufficiently different that

they cannot be done efficiently with 40 uniquely defined pins. Consequently, the 80C86A is equipped with a strap pin (MN/MX) which defines the system configuration. The definition of a certain subset of the pins changes dependent on the condition of the strap pin. When MN/MX pin is strapped to GND, the 80C86A treats pins 24 through 31 in maximum mode. An 82C88 bus controller interprets status information coded into $\overline{S}_0, \overline{S}_1, \overline{S}_2$ to generate bus timing and control signals compatible with the MULTIBUS® architecture. When the MN/MX pin is strapped to VCC, the 80C86A generates bus control signals itself on pins 24 through 31, as shown in parentheses in Figure 2. Examples of minimum mode and maximum mode systems are shown in Figure 4.

BUS OPERATION

The 80C86A has a combined address and data bus commonly referred to as a time multiplexed bus. This technique provides the most efficient use of pins on the processor. This "local bus" can be buffered directly and used throughout the system with address latching provided on memory and I/O modules. In addition, the bus can also be demultiplexed at the processor with a single set of address latches if a standard non-multiplexed bus is desired for the system.

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T₁, T₂, T₃ and T₄ (see Figure 5). The address is emitted from the processor during T₁ and data transfer occurs on the bus during T₃ and T₄. T₂ is used primarily for changing the direction of the bus during read operations. In the event that a "NOT READY" indication is given by the addressed device, "Wait" states (T_W) are inserted between T₃ and T₄. Each inserted "Wait" state is of the same duration as a CLK cycle. Periods can occur between 80C86A bus cycles. These are referred to as "Idle" states (T_i) or inactive CLK cycles. The processor uses these cycles for internal housekeeping.

During T₁ of any bus cycle the ALE (Address Latch Enable) signal is emitted (by either the processor or the 82C88 bus controller, depending on the MN/MX strap). At the trailing edge of this pulse, a valid address and certain status information for the cycle may be latched.

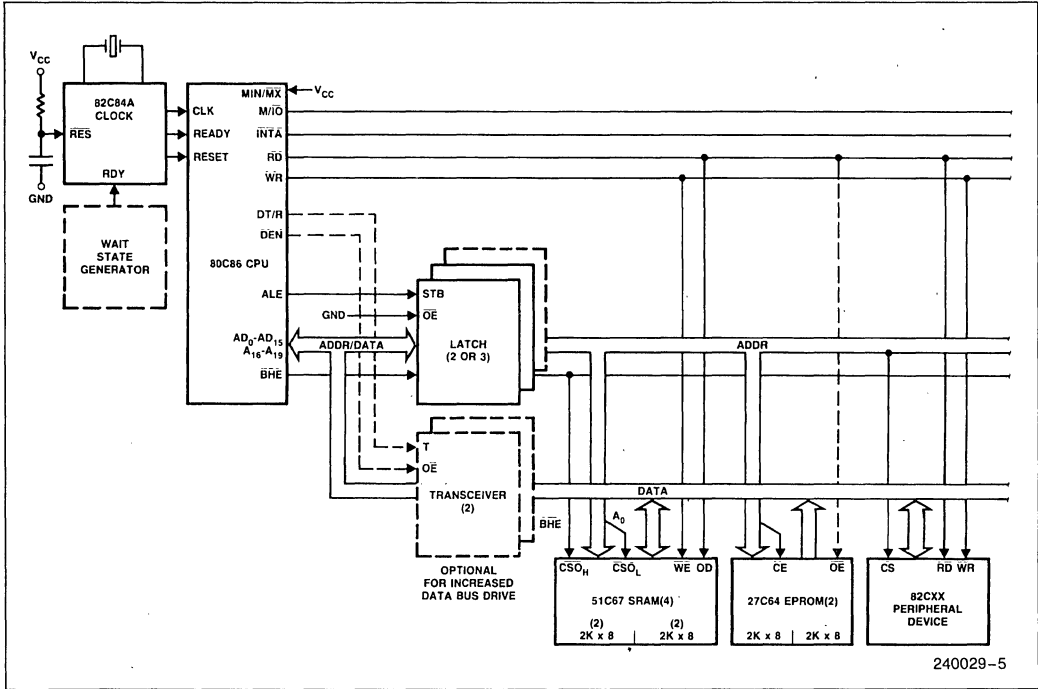


Figure 4a. Minimum Mode iAPX 80C86A Typical Configuration

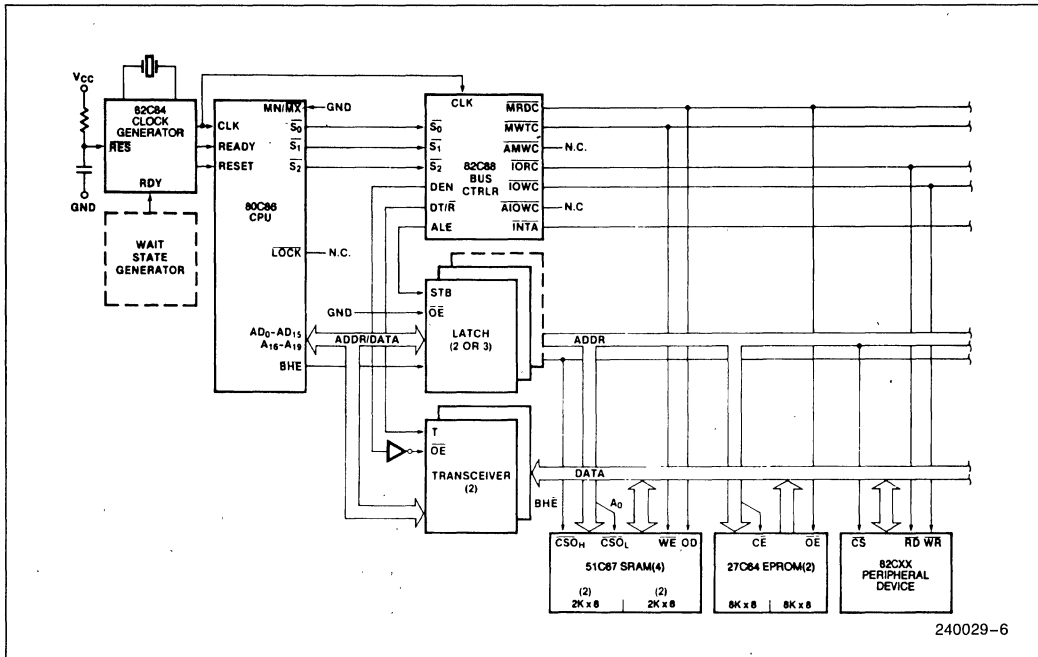


Figure 4b. Maximum Mode 80C86A Typical Configuration

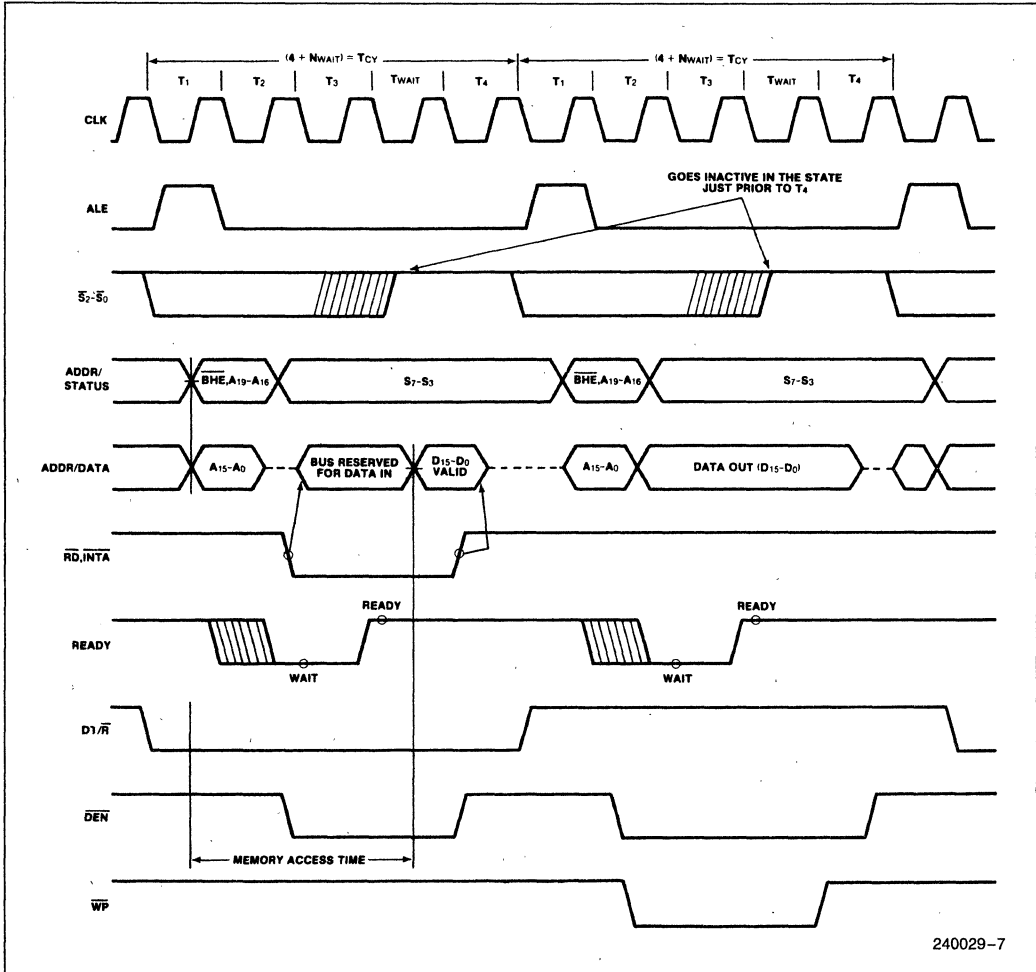


Figure 5. Basic System Timing

Status bits \bar{S}_0 , \bar{S}_1 , and \bar{S}_2 are used, in maximum mode, by the bus controller to identify the type of bus transaction according to the following table:

\bar{S}_2	\bar{S}_1	\bar{S}_0	Characteristics
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

therefore valid during T₂ through T₄. S₃ and S₄ indicate which segment register (see Instruction Set description) was used for this bus cycle in forming the address, according to the following table:

S ₄	S ₃	Characteristics
0 (LOW)	0	Alternate Data (extra segment)
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data

S₅ is a reflection of the PSW interrupt enable bit. S₆=0 and S₇ is a spare status pin.

Status bits S₃ through S₇ are multiplexed with high-order address bits and the BHE signal, and are

I/O ADDRESSING

In the 80C86A, I/O operations can address up to a maximum of 64k I/O byte registers or 32k I/O word registers. The I/O address appears in the same format as the memory address on bus lines A₁₅-A₀. The address lines A₁₉-A₁₆ are zero in I/O operations. The variable I/O instructions which use register DX as a pointer have full address capability while the direct I/O instructions directly address one or two of the 256 I/O byte locations in page 0 of the I/O address space.

I/O ports are addressed in the same manner as memory locations. Even addressed bytes are transferred on the D₇-D₀ bus lines and odd addressed bytes on D₁₅-D₈. Care must be taken to assure that each register within an 8-bit peripheral located on the lower portion of the bus be addressed as even.

EXTERNAL INTERFACE

PROCESSOR RESET AND INITIALIZATION

Processor initialization or start up is accomplished with activation (HIGH) of the RESET pin. The 80C86A RESET is required to be HIGH for four or more CLK cycles. The 80C86A will terminate operations on the high-going edge of RESET and will remain dormant as long as RESET is HIGH. The low-going transition of RESET triggers an internal reset sequence for approximately 7 CLK cycles. After this interval the 80C86A operates normally beginning with the instruction in absolute location FFFF0H (see Figure 3b). The details of this operation are specified in the Instruction Set description of the MCS[®]-86 Family User's Manual. The RESET input is internally synchronized to the processor clock. At

initialization the HIGH-to-LOW transition of RESET must occur no sooner than 50 μs after power-up, to allow complete initialization of the 80C86A.

NMI asserted prior to the 2nd clock after the end of RESET will not be honored. If NMI is asserted after that point and during the internal reset sequence, the processor may execute one instruction before responding to the interrupt. A hold request active immediately after RESET will be honored before the first instruction fetch.

All 3-state outputs float to 3-state OFF⁽¹⁾ during RESET. Status is active in the idle state for the first clock after RESET becomes active and then floats to 3-state OFF⁽¹⁾. ALE and HLDA are driven low.

NOTE:

1. See the section on Bus Hold Circuitry.

BUS HOLD CIRCUITRY

To avoid high current conditions caused by floating inputs to CMOS devices and eliminate the need for pull-up/down resistors, "bus-hold" circuitry has been used on the 80C86A pins 2-16, 26-32, and 34-39 (Figures 6a, 6b). These circuits will maintain the last valid logic state if no driving source is present (i.e. an unconnected pin or a driving source which goes to a high impedance state). To override the "bus hold" circuits, an external driver must be capable of supplying 350 μA minimum sink or source current at valid input voltage levels. Since this "bus hold" circuitry is active and not a "resistive" type element, the associated power supply current is negligible and power dissipation is significantly reduced when compared to the use of passive pull-up resistors.

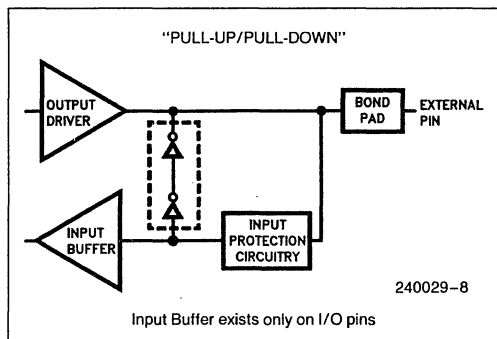


Figure 6a. Bus hold circuitry pin 2-16, 34-39.

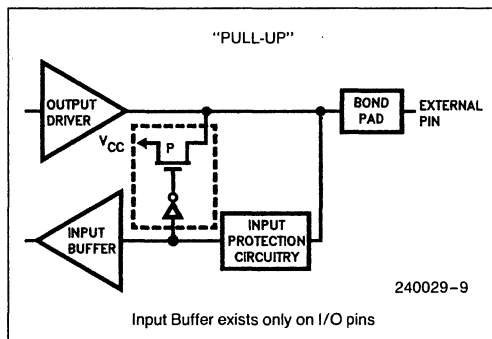


Figure 6b. Bus hold circuitry pin 26-32.

INTERRUPT OPERATIONS

Interrupt operations fall into two classes; software or hardware initiated. The software initiated interrupts and software aspects of hardware interrupts are specified in the Instruction Set description. Hardware interrupts can be classified as non-maskable or maskable.

Interrupts result in a transfer of control to a new program location. A 256-element table containing address pointers to the interrupt service program locations resides in absolute locations 0 through 3FFH (see Figure 3b), which are reserved for this purpose. Each element in the table is 4 bytes in size and corresponds to an interrupt "type". An interrupting device supplies an 8-bit type number, during the interrupt acknowledge sequence, which is used to "vector" through the appropriate element to the new interrupt service program location.

NON-MASKABLE INTERRUPT (NMI)

The processor provides a single non-maskable interrupt pin (NMI) which has higher priority than the maskable interrupt request pin (INTR). A typical use would be to activate a power failure routine. The NMI is edge-triggered on a LOW-to-HIGH transition. The activation of this pin causes a type 2 interrupt. (See Instruction Set description.) NMI is required to have a duration in the HIGH state of greater than two CLK cycles, but is not required to be synchronized to the clock. Any high-going transition of NMI is latched on-chip and will be serviced at the end of the current instruction or between whole moves of a block-type instruction. Worst case response to NMI would be for multiply, divide and variable shift instructions. There is no specification on the occurrence of the low-going edge; it may occur before, during, or after the servicing of NMI. Another high-going edge triggers another response if it occurs after the start of the NMI procedure. The signal must be free of logical spikes in general and be free of bounces on the low-going edge to avoid triggering extraneous responses.

MASKABLE INTERRUPT (INTR)

The 80C86A provides a single interrupt request input (INTR) which can be masked internally by software

with the resetting of the interrupt enable FLAG status bit. The interrupt request signal is level triggered. It is internally synchronized during each clock cycle on the high-going edge of CLK. To be responded to, INTR must be present (HIGH) during the clock period preceding the end of the current instruction or the end of a whole move for a block-type instruction. During the interrupt response sequence further interrupts are disabled. The enable bit is reset as part of the response to any interrupt (INTR, NMI, software interrupt or single-step), although the FLAGS register which is automatically pushed onto the stack reflects the state of the processor prior to the interrupt. Until the old FLAGS register is restored the enable bit will be zero unless specifically set by an instruction.

During the response sequence (Figure 7) the processor executes two successive (back-to-back) interrupt acknowledge cycles. The 80C86A emits the LOCK signal from T₂ of the first bus cycle until T₂ of the second. A local bus "hold" request will not be honored until the end of the second bus cycle. In the second bus cycle a byte is fetched from the external interrupt system (e.g., 82C59 PIC) which identifies the source (type) of the interrupt. This byte is multiplied by four and used as a pointer into the interrupt vector lookup table. An INTR signal left HIGH will be continually responded to within the limitations of the enable bit and sample period. The INTERRUPT RETURN instruction includes a FLAGS pop which returns the status of the original interrupt enable bit when it restores the FLAGS.

HALT

When a software "HALT" instruction is executed the processor indicates that it is entering the "HALT" state in one of two ways depending upon which mode is strapped. In minimum mode, the processor issues one ALE with no qualifying bus control signals. In Maximum Mode, the processor issues appropriate HALT status on \overline{S}_2 , \overline{S}_1 and \overline{S}_0 and the 82C88 bus controller issues one ALE. The 80C86A will not leave the "HALT" state when a local bus "hold" is entered while in "HALT". In this case, the processor reissues the HALT indicator. An interrupt request or RESET will force the 80C86A out of the "HALT" state.

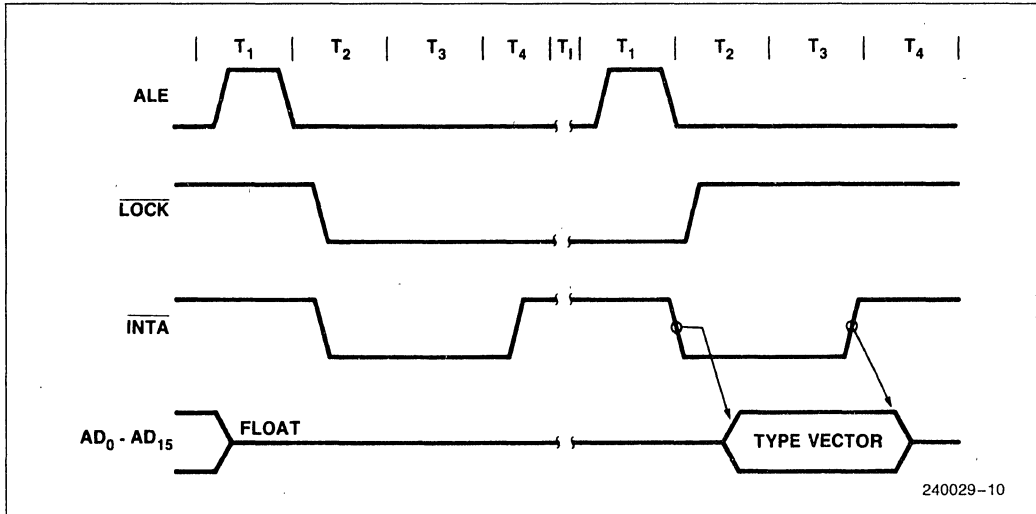


Figure 7. Interrupt Acknowledge Sequence

READ/MODIFY/WRITE (SEMAPHORE) OPERATIONS VIA LOCK

The $\overline{\text{LOCK}}$ status information is provided by the processor when directly consecutive bus cycles are required during the execution of an instruction. This provides the processor with the capability of performing read/modify/write operations on memory (via the Exchange Register With Memory instruction, for example) without the possibility of another system bus master receiving intervening memory cycles. This is useful in multiprocessor system configurations to accomplish "test and set lock" operations. The $\overline{\text{LOCK}}$ signal is activated (forced LOW) in the clock cycle following the one in which the software "LOCK" prefix instruction is decoded by the EU. It is deactivated at the end of the last bus cycle of the instruction following the "LOCK" prefix instruction. While $\overline{\text{LOCK}}$ is active a request on a RQ/GT pin will be recorded and then honored at the end of the LOCK.

EXTERNAL SYNCHRONIZATION VIA TEST

As an alternative to the interrupts and general I/O capabilities, the 80C86A provides a single software-testable input known as the TEST signal. At any time the program may execute a WAIT instruction. If at that time the TEST signal is inactive (HIGH), pro-

gram execution becomes suspended while the processor waits for $\overline{\text{TEST}}$ to become active. It must remain active for at least 5 CLK cycles. The WAIT instruction is re-executed repeatedly until that time. This activity does not consume bus cycles. The processor remains in an idle state while waiting. All 80C86A drivers go to 3-state OFF if bus "Hold" is entered. If interrupts are enabled, they may occur while the processor is waiting. When this occurs the processor fetches the WAIT instruction one extra time, processes the interrupt, and then re-fetches and re-executes the WAIT instruction upon returning from the interrupt.

BASIC SYSTEM TIMING

Typical system configurations for the processor operating in minimum mode and in maximum mode are shown in Figures 4a and 4b, respectively. In minimum mode, the MN/MX pin is strapped to V_{CC} and the processor emits bus control signals in a manner similar to the 8085. In maximum mode, the MN/MX pin is strapped to V_{SS} and the processor emits coded status information which the 82C88 bus controller uses to generate MULTIBUS compatible bus control signals. Figure 5 illustrates the signal timing relationships.

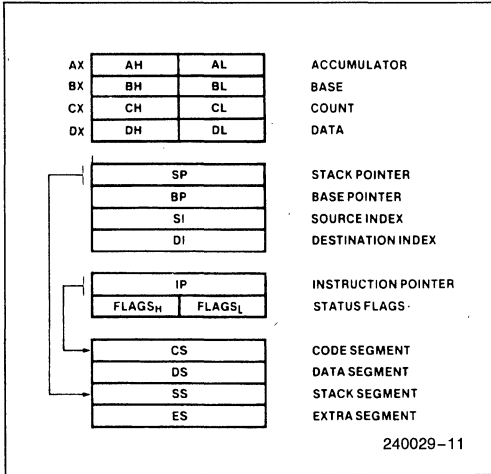


Figure 8. 80C86A Register Model

SYSTEM TIMING—MINIMUM SYSTEM

The read cycle begins in T_1 with the assertion of the Address Latch Enable (ALE) signal. The trailing (low-going) edge of this signal is used to latch the address information, which is valid on the local bus at this time, into a latch. The BHE and A_0 signals address the low, high, or both bytes. From T_1 to T_4 the M/\overline{IO} signal indicates a memory or I/O operation. At T_2 the address is removed from the local bus and the bus goes to a high impedance state. The read control signal is also asserted at T_2 . The read (\overline{RD}) signal causes the addressed device to enable its data bus drivers to the local bus. Some time later valid data will be available on the bus and the addressed device will drive the $READY$ line HIGH. When the processor returns the read signal to a HIGH level, the addressed device will again 3-state its bus drivers. If a transceiver is required to buffer the 80C86A local bus, signals DT/\overline{R} and \overline{DEN} are provided by the 80C86A.

A write cycle also begins with the assertion of ALE and the emission of the address. The M/\overline{IO} signal is again asserted to indicate a memory or I/O write operation. In the T_2 immediately following the address emission the processor emits the data to be written into the addressed location. This data remains valid until the middle of T_4 . During T_2 , T_3 , and T_W the processor asserts the write control signal. The write (\overline{WR}) signal becomes active at the beginning of T_2 as opposed to the read which is delayed somewhat into T_2 to provide time for the bus to float.

The BHE and A_0 signals are used to select the proper byte(s) of the memory/I/O word to be read or written according to the following table:

BHE	A_0	Characteristics
0	0	Whole word
0	1	Upper byte from/to odd address
1	0	Lower byte from/to even address
1	1	None

I/O ports are addressed in the same manner as memory location. Even addressed bytes are transferred on the D_7-D_0 bus lines and odd addressed bytes on $D_{15}-D_8$.

The basic difference between the interrupt acknowledge cycle and a read cycle is that the interrupt acknowledge signal (\overline{INTA}) is asserted in place of the read (\overline{RD}) signal and the address bus is floated. (See Figure 7.) In the second of two successive \overline{INTA} cycles, a byte of information is read from bus lines D_7-D_0 as supplied by the interrupt system logic (i.e., 82C59A Priority Interrupt Controller). This byte identifies the source (type) of the interrupt. It is multiplied by four and used as a pointer into an interrupt vector lookup table, as described earlier.

BUS TIMING—MEDIUM SIZE SYSTEMS

For medium size systems the MN/\overline{MX} pin is connected to V_{SS} and the 82C88 Bus Controller is added to the system as well as a latch for latching the system address, and a transceiver to allow for bus loading greater than the 80C86A is capable of handling. Signals ALE, DEN, and DT/\overline{R} are generated by the 82C88 instead of the processor in this configuration although their timing remains relatively the same. The 80C86A status outputs ($\overline{S_2}$, $\overline{S_1}$, and $\overline{S_0}$) provide type-of-cycle information and become 82C88 inputs. This bus cycle information specifies read (code, data, or I/O), write (data or I/O), interrupt acknowledge, or software halt. The 82C88 thus issues control signals specifying memory read or write, I/O read or write, or interrupt acknowledge. The 82C88 provides two types of write strobes, normal and advanced, to be applied as required. The normal write strobes have data valid at the leading edge of write. The advanced write strobes have the same timing as read strobes, and hence data isn't valid at the leading edge of write. The transceiver receives the usual T and OE inputs from the 82C88 DT/\overline{R} and DEN.

The pointer into the interrupt vector table, which is passed during the second \overline{INTA} cycle, can derive from an 82C59A located on either the local bus or the system bus. If the master 82C59A Priority Interrupt Controller is positioned on the local bus, a TTL gate is required to disable the transceiver when reading from the master 82C59A during the interrupt acknowledge sequence and software "poll".

ABSOLUTE MAXIMUM RATINGS*

Supply Voltage
 (With respect to ground) -0.5 to 7.0V
 Input Voltage Applied
 (w.r.t. ground) -0.5 to $V_{CC} + 0.5V$
 Output Voltage Applied
 (w.r.t. ground) -0.5 to $V_{CC} + 0.5V$
 Power Dissipation 1.0W
 Storage Temperature -65°C to 150°C
 Ambient Temperature Under Bias 0°C to 70°C

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

($T_A = 0^\circ C$ to $70^\circ C$, $V_{CC} = 5V \pm 5\%$)

Symbol	Parameter	80C86A-2		Units	Test Conditions
		Min	Max		
V_{IL}	Input Low Voltage	-0.5	+0.8	V	
V_{IH}	Input High Voltage (All inputs except clock)	2.0		V	
V_{CH}	Clock Input High Voltage	$V_{CC} - 0.8$		V	
V_{OL}	Output Low Voltage		0.45	V	$I_{OL} = 2.5\text{ mA}$
V_{OH}	Output High Voltage	3.0 $V_{CC} - 0.4$		V	$I_{OH} = -2.5\text{ mA}$ $I_{OH} = -100\text{ }\mu\text{A}$
I_{CC}	Power Supply Current		10 mA/MHz		$V_{IL} = \text{GND}$, $V_{IH} = V_{CC}$
I_{CCS}	Standby Supply Current		500	μA	$V_{IN} = V_{CC}$ or GND Outputs Unloaded CLK = GND or V_{CC}
I_{LI}	Input Leakage Current		± 1.0	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{BHL}	Input Leakage Current (Bus Hold Low)	50	400	μA	$V_{IN} = 0.8V$ (Note 4)
I_{BHH}	Input Leakage Current (Bus Hold High)	-50	-400	μA	$V_{IN} = 3.0V$ (Note 5)
I_{BHLO}	Bus Hold Low Overdrive		600	μA	(Note 2)
I_{BHHO}	Bus Hold High Overdrive		-600	μA	(Note 3)
I_{LO}	Output Leakage Current		± 10	μA	$V_{OUT} = \text{GND}$ or V_{CC}
C_{IN}	Capacitance of Input Buffer (All inputs except AD_0 - AD_{15} , RQ/GT)		5	pF	(Note 1)
C_{IO}	Capacitance of I/O Buffer (AD_0 - AD_{15} , RQ/GT)		20	pF	(Note 1)
C_{OUT}	Output Capacitance		15	pF	(Note 1)

NOTES:

1. Characterization conditions are a) Frequency = 1 MHz; b) Unmeasured pins at GND; c) V_{IN} at +5.0V or GND.
2. An external driver must source at least I_{BHLO} to switch this node from LOW to HIGH.
3. An external driver must sink at least I_{BHHO} to switch this node from HIGH to LOW.
4. Test Condition is to lower V_{IN} to GND and then raise V_{IN} to 0.8V on pins 2-16 & 34-39.
5. Test Condition is to raise V_{IN} to V_{CC} and then lower V_{IN} to 3.0V on pins 2-16, 26-32 & 34-39.

A.C. CHARACTERISTICS

 (T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

Symbol	Parameter	80C86A-2		Units	Test Conditions
		Min	Max		
TCLCL	CLK Cycle Period	125	D.C.	ns	
TCLCH	CLK Low Time	68		ns	
TCHCL	CLK High Time	44		ns	
TCH1CH2	CLK Rise Time		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	20		ns	
TCLDX	Data in Hold Time	10		ns	
TR1VCL	RDY Setup Time into 82C84A (Notes 1, 2)	35		ns	
TCLR1X	RDY Hold Time into 82C84A (Notes 1, 2)	0		ns	
TRYHCH	READY Setup Time into 80C86A	68		ns	
TCHRYX	READY Hold Time into 80C86A	20		ns	
TRYLCL	READY Inactive to CLK (Note 3)	-8		ns	
THVCH	HOLD Setup Time	20		ns	
TINVCH	INTR, NMI, $\overline{\text{TEST}}$ Setup Time (Note 2)	15		ns	
TILIH	Input Rise Time (Except CLK)		15	ns	
TIHIL	Input Fall Time (Except CLK)		15	ns	From 2.0V to 0.8V

A.C. CHARACTERISTICS (Continued)

 (T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

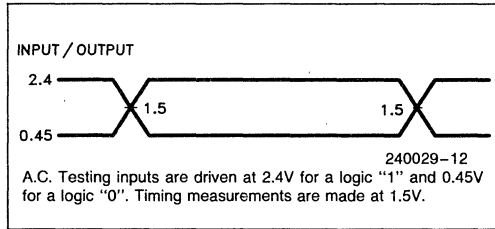
Timing Responses

Symbol	Parameter	80C86A-2		Units	Test Conditions	
		Min	Max			
TCLAV	Address Valid Delay	10	60	ns		
TCLAX	Address Hold Time	10		ns		
TCLAZ	Address Float Delay	TCLAX	50	ns		
TLHLL	ALE Width	TCLCH – 10		ns		
TCLLH	ALE Active Delay		50	ns		
TCHLL	ALE Inactive Delay		55	ns		
TLLAX	Address Hold Time to ALE Inactive	TCHCL – 10		ns		
TCLDV	Data Valid Delay	10	60	ns		
TCHDX	Data Hold Time	10		ns		
TWHDX	Data Hold Time After WR	TCLCH – 30		ns		
TCVCTV	Control Active Delay 1	10	70	ns		
TCHCTV	Control Active Delay 2	10	60	ns		
TCVCTX	Control Inactive Delay	10	70	ns		
TAZRL	Address Float to READ Active	0		ns		
TCLRL	\overline{RD} Active Delay	10	100	ns		
TCLRHR	\overline{RD} Inactive Delay	10	80	ns		
TRHAV	\overline{RD} Inactive to Next Address Active	TCLCL – 40		ns		
TCLHAV	HLDA Valid Delay	10	100	ns		
TRLRH	\overline{RD} Width	2TCLCL – 50		ns		
TWLWH	\overline{WR} Width	2TCLCL – 40		ns		
TAVAL	Address Valid to ALE Low	TCLCH – 40		ns		
TOLOH	Output Rise Time		15	ns		From 0.8V to 2.0V
TOHOL	Output Fall Time		15	ns		From 2.0V to 0.8V

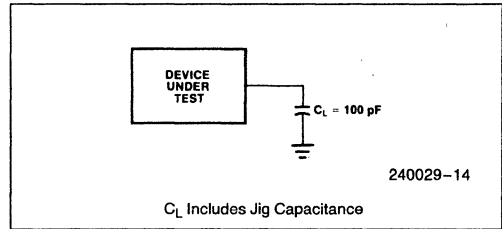
NOTES:

- Signal at 82C84A shown for reference only. See 82C84A data sheet for the most recent specifications.
- Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
- Applies only to T2 state. (8 ns into T3).

A.C. TESTING INPUT, OUTPUT WAVEFORM

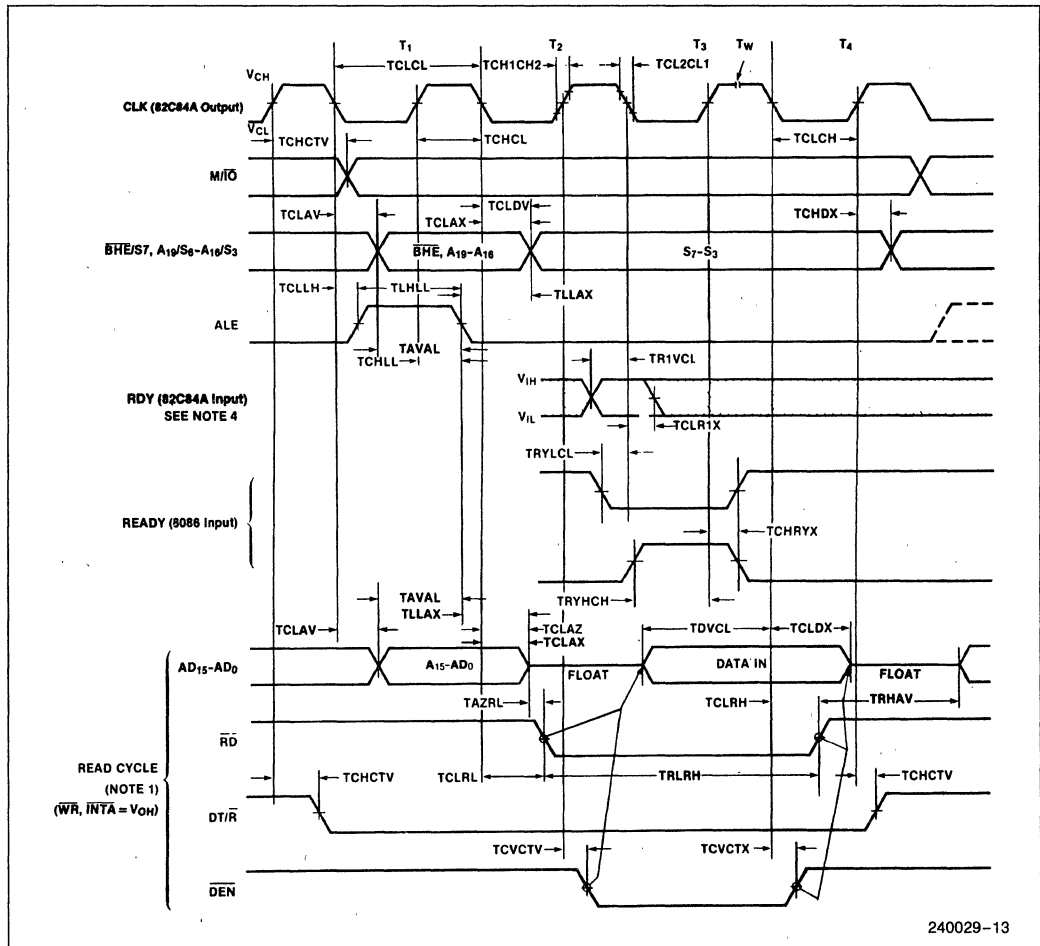


A.C. TESTING LOAD CIRCUIT

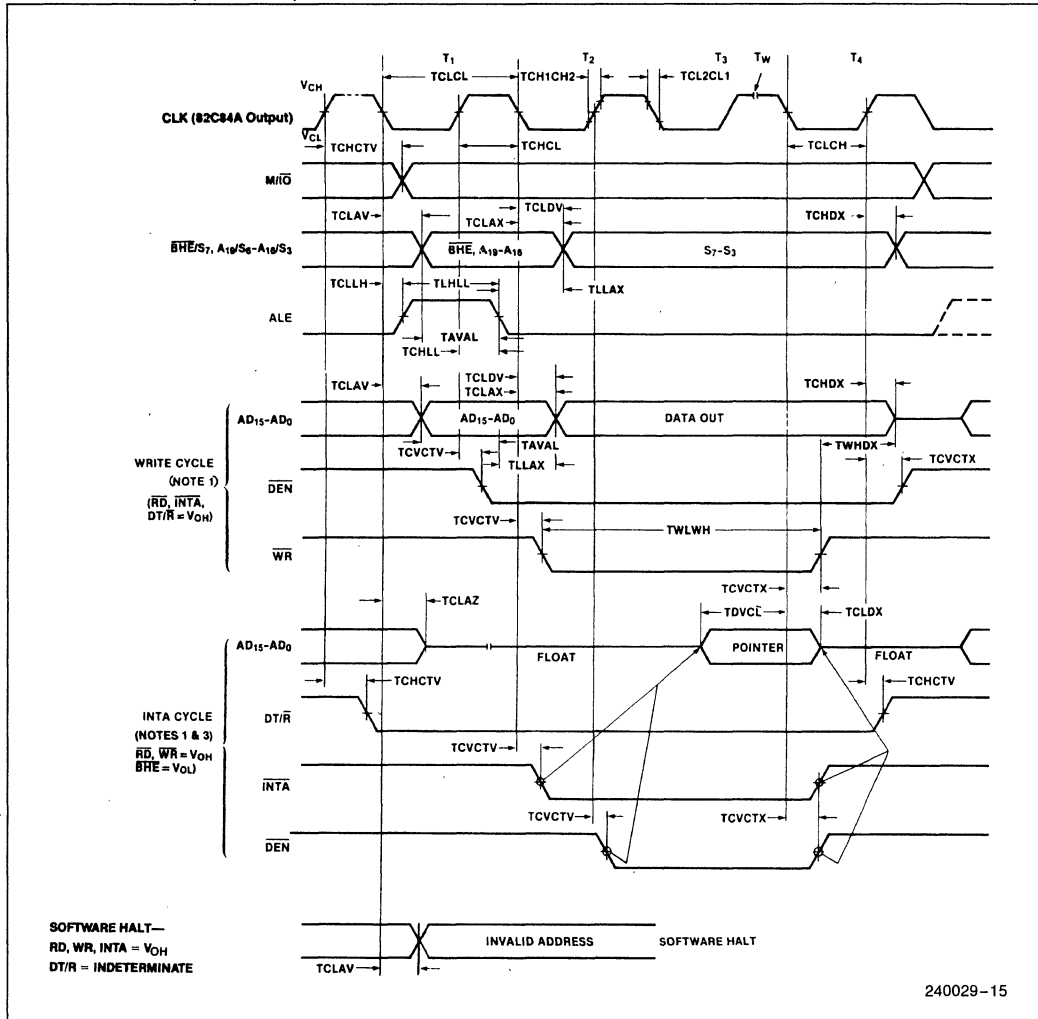


WAVEFORMS

MINIMUM MODE



WAVEFORMS (Continued)

MINIMUM MODE (Continued)


240029-15

NOTES:

1. All output timing measurements are made at 1.5V unless otherwise noted.
2. RDY is sampled near the end of T_2 , T_3 , T_W to determine if T_W machines states are to be inserted.
3. Two INTA cycles run back-to-back. The 80C86A local ADDR/DATA BUS is floating during both INTA cycles. Control signals shown for second INTA cycle.
4. Signals at 82C84A are shown for reference only.

A.C. CHARACTERISTICS
**MAX MODE SYSTEM (USING 82C88 BUS CONTROLLER)
TIMING REQUIREMENTS**

Symbol	Parameter	80C86A-2		Units	Test Conditions	
		Min	Max			
TCLCL	CLK Cycle Period	125	D.C.	ns		
TCLCH	CLK Low Time	68		ns		
TCHCL	CLK High Time	44		ns		
TCH1CH2	CLK Rise Time		10	ns	From 1.0V to 3.5V	
TCL2CL1	CLK Fall Time		10	ns	From 3.5V to 1.0V	
TDVCL	Data in Setup Time	20		ns		
TCLDX	Data in Hold Time	10		ns		
TR1VCL	RDY Setup Time into 82C84A (Notes 1, 2)	35		ns		
TCLR1X	RDY Hold Time into 82C84A (Notes 1, 2)	0		ns		
TRYHCH	READY Setup Time into 80C86A	68		ns		
TCHRYX	READY Hold Time into 80C86A	20		ns		
TRYLCL	READY Inactive to CLK (Note 4)	-8		ns		
TINVCH	Setup Time for Recognition (INTR, NMI, TEST) (Note 2)	15		ns		
TGVCH	$\overline{RQ}/\overline{GT}$ Setup Time	15		ns		
TCHGX	\overline{RQ} Hold Time into 80C86A	30		ns		
TILIH	Input Rise Time (Except CLK) (Note 5)		15	ns		From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK) (Note 5)		15	ns		From 2.0V to 0.8V

A.C. CHARACTERISTICS (Continued)

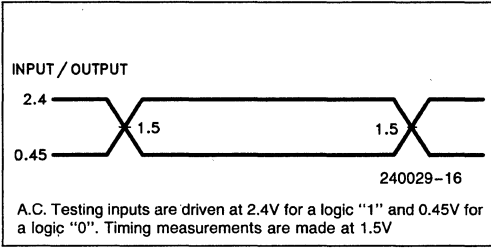
TIMING RESPONSES

Symbol	Parameter	80C86A-2		Units	Test Conditions
		Min	Max		
TCLML	Command Active Delay (Note 1)	5	35	ns	
TCLMH	Command Inactive Delay (Note 1)	5	35	ns	
TRYHSH	READY Active to Status Passive (Note 3)		65	ns	
TCHSV	Status Active Delay	10	60	ns	
TCLSH	Status Inactive Delay	10	70	ns	
TCLAV	Address Valid Delay	10	60	ns	
TCLAX	Address Hold Time	10		ns	
TCLAZ	Address Float Delay	TCLAX	50	ns	
TSVLH	Status Valid to ALE High (Note 1)		20	ns	
TSVMCH	Status Valid to MCE High (Note 1)		30	ns	
TCLLH	CLK Low to ALE Valid (Note 1)		20	ns	
TCLMCH	CLK Low to MCE High (Note 1)		25	ns	
TCHLL	ALE Inactive Delay (Note 1)	4	18	ns	
TCLDV	Data Valid Delay	10	60	ns	
TCHDX	Data Hold Time	10		ns	
TCVNV	Control Active Delay (Note 1)	5	45	ns	
TCVNX	Control Inactive Delay (Note 1)	10	45	ns	
TAZRL	Address Float to Read Active	0		ns	
TCLRL	RD Active Delay	10	100	ns	
TCLRH	RD Inactive Delay	10	80	ns	
TRHAV	RD Inactive to Next Address Active	TCLCL - 40		ns	
TCHDTL	Direction Control Active Delay (Note 1)		50	ns	
TCHDTH	Direction Control Inactive Delay (Note 1)		30	ns	
TCLGL	GT Active Delay	0	50	ns	
TCLGH	GT Inactive Delay	0	50	ns	
TRLRH	RD Width	2TCLCL - 50		ns	
TOLOH	Output Rise Time		15	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		15	ns	From 2.0V to 0.8V

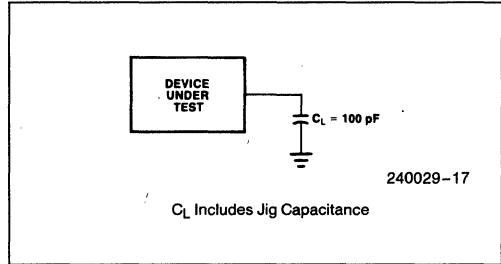
NOTES:

- Signal at 82C84A or 82C88 shown for reference only. See 82C84A and 82C88 for the most recent specifications.
- Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
- Applies only to T3 and wait states.
- Applies only to T2 state (8 ns into T3).
- These parameters are characterized and not 100% tested.

A.C. TESTING INPUT, OUTPUT WAVEFORM

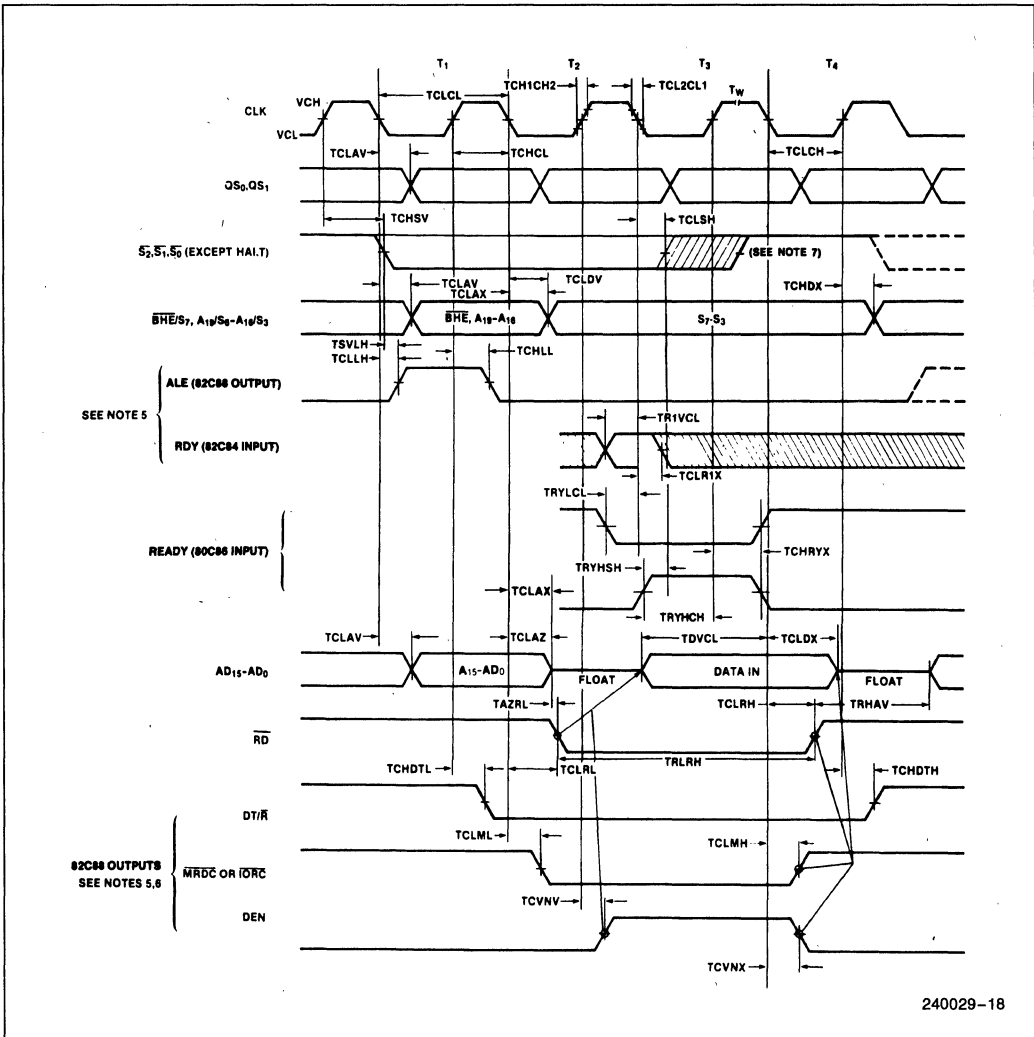


A.C. TESTING LOAD CIRCUIT



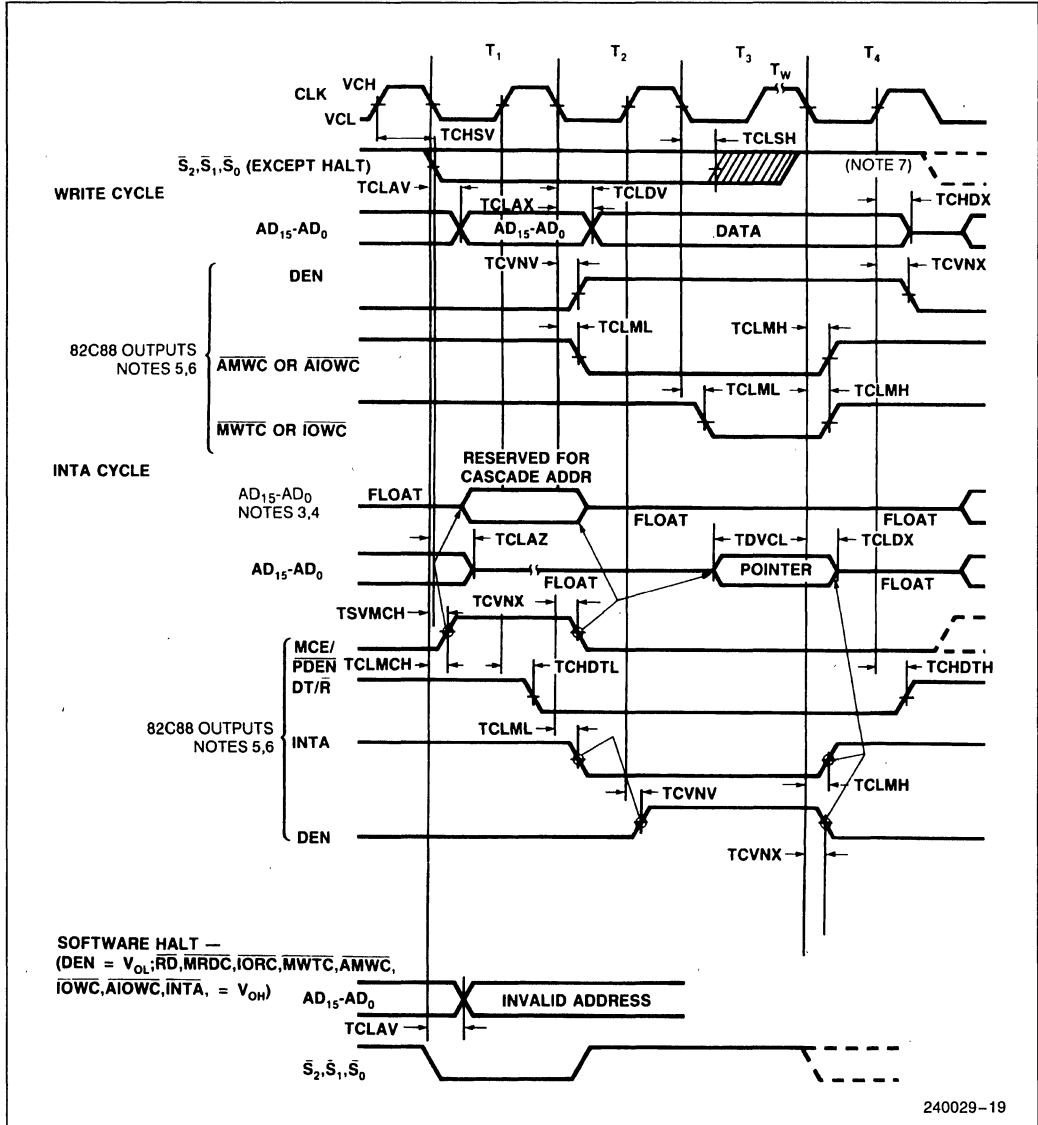
WAVEFORMS

MAXIMUM MODE



WAVEFORMS (Continued)

MAXIMUM MODE (Continued)



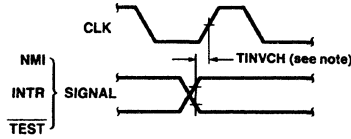
240029-19

NOTES:

1. All timing measurements are made at 1.5V unless otherwise noted.
2. RDY is sampled near the end of T₂, T₃, T_W to determine if T_W machines states are to be inserted.
3. Cascade address is valid between first and second INTA cycle.
4. Two INTA cycles run back-to-back. The 80C86A local ADDR/DATA BUS is floating during both INTA cycles. Control for pointer address is shown for second INTA cycle.
5. Signals at 82C84A or 82C88 are shown for reference only.
6. The issuance of the 82C88 command and control signals (MRDC, MWTC, AMWC, IORC, IOWC, AIOWC, INTA and DEN) lags the active high 82C88 CEN.
7. Status inactive in state just prior to T₄.

WAVEFORMS (Continued)

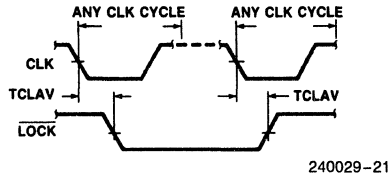
ASYNCHRONOUS SIGNAL RECOGNITION



240029-20

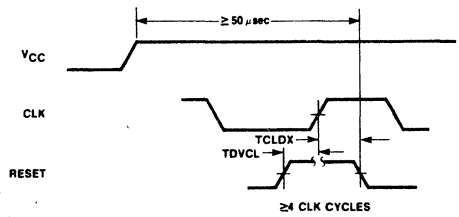
NOTE: Setup requirements for asynchronous signals only to guarantee recognition at next CLK.

BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)



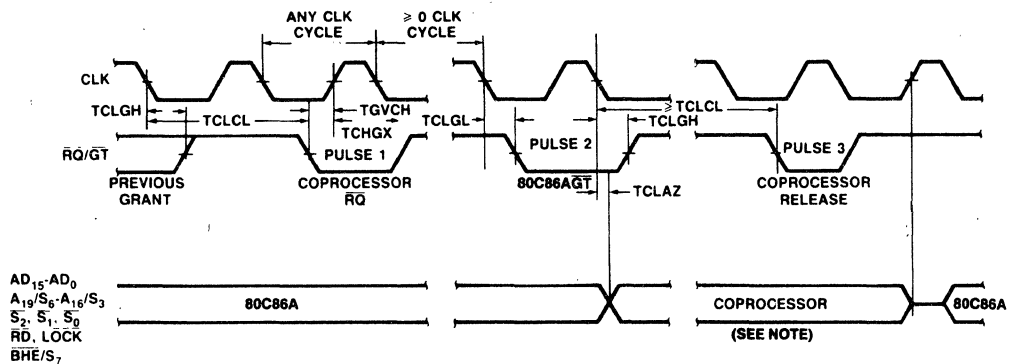
240029-21

RESET TIMING



240029-22

REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)

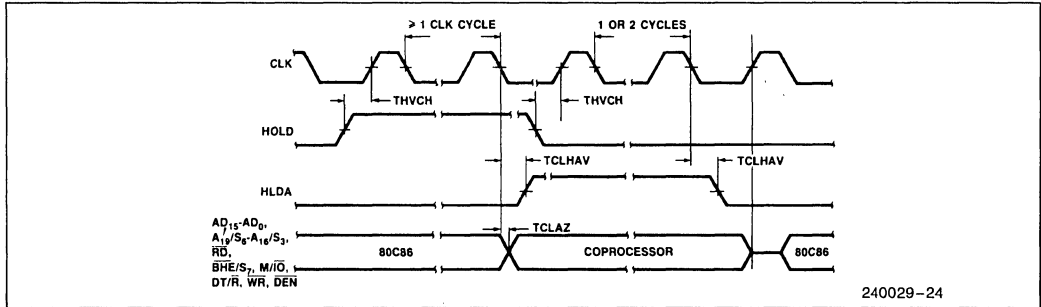


240029-23

NOTE: The coprocessor may not drive the buses outside the region shown without risking contention.

WAVEFORMS (Continued)

HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)



240029-24

Table 2. Instruction Set Summary

Mnemonic and Description	Instruction Code			
DATA TRANSFER				
MOV = Move:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Register/Memory to/from Register**	1 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate to Register	1 0 1 1 w reg	data	data if w = 1	
Memory to Accumulator	1 0 1 0 0 0 w	addr-low	addr-high	
Accumulator to Memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/Memory to Segment Register**	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment Register to Register/Memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		
PUSH = Push:				
Register/Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m		
Register	0 1 0 1 0 reg			
Segment Register	0 0 0 reg 1 1 0			
POP = Pop:				
Register/Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m		
Register	0 1 0 1 1 reg			
Segment Register	0 0 0 reg 1 1 1			
XCHG = Exchange:				
Register/Memory with Register	1 0 0 0 0 1 1 w	mod reg r/m		
Register with Accumulator	1 0 0 1 0 reg			
IN = Input from:				
Fixed Port	1 1 1 0 0 1 0 w	port		
Variable Port	1 1 1 0 1 1 0 w			
OUT = Output to:				
Fixed Port	1 1 1 0 0 1 1 w	port		
Variable Port	1 1 1 0 1 1 1 w			
XLAT = Translate Byte to AL	1 1 0 1 0 1 1 1			
LEA = Load EA to Register	1 0 0 0 1 1 0 1	mod reg r/m		
LDS = Load Pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m		
LES = Load Pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m		
LAHF = Load AH with Flags	1 0 0 1 1 1 1 1			
SAHF = Store AH into Flags	1 0 0 1 1 1 1 0			
PUSHF = Push Flags	1 0 0 1 1 1 0 0			
POPF = Pop Flags	1 0 0 1 1 1 0 1			

Table 2. Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code			
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
ARITHMETIC				
ADD = Add:				
Reg./Memory with Register to Either	0 0 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s w = 0 1
Immediate to Accumulator	0 0 0 0 0 1 0 w	data	data if w = 1	
ADC = Add with Carry:				
Reg./Memory with Register to Either	0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s w = 0 1
Immediate to Accumulator	0 0 0 1 0 1 0 w	data	data if w = 1	
INC = Increment:				
Register/Memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m		
Register	0 1 0 0 0 reg			
AAA = ASCII Adjust for Add	0 0 1 1 0 1 1 1			
DAA = Decimal Adjust for Add	0 0 1 0 0 1 1 1			
SUB = Subtract:				
Reg./Memory and Register to Either	0 0 1 0 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s w = 0 1
Immediate from Accumulator	0 0 1 0 1 1 0 w	data	data if w = 1	
SBB = Subtract with Borrow				
Reg./Memory and Register to Either	0 0 0 1 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s w = 0 1
Immediate from Accumulator	0 0 0 1 1 1 0 w	data	data if w = 1	
DEC = Decrement:				
Register/Memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m		
Register	0 1 0 0 1 reg			
NEG = Change Sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		
CMP = Compare:				
Register/Memory and Register	0 0 1 1 1 0 d w	mod reg r/m		
Immediate with Register/Memory	1 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s w = 0 1
Immediate with Accumulator	0 0 1 1 1 1 0 w	data	data if w = 1	
AAS = ASCII Adjust for Subtract	0 0 1 1 1 1 1 1			
DAS = Decimal Adjust for Subtract	0 0 1 0 1 1 1 1			
MUL = Multiply (Unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m		
IMUL = Integer Multiply (Signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m		
AAM = ASCII Adjust for Multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		
DIV = Divide (Unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m		
IDIV = Integer Divide (Signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m		
AAD = ASCII Adjust for Divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		
CBW = Convert Byte to Word	1 0 0 1 1 0 0 0			
CWD = Convert Word to Double Word	1 0 0 1 1 0 0 1			

8086/8088 Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code			
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
LOGIC				
NOT = Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m		
SHL/SAL = Shift Logical/Arithmetic Left	1 1 0 1 0 0 v w	mod 1 0 0 r/m		
SHR = Shift Logical Right	1 1 0 1 0 0 v w	mod 1 0 1 r/m		
SAR = Shift Arithmetic Right	1 1 0 1 0 0 v w	mod 1 1 1 r/m		
ROL = Rotate Left	1 1 0 1 0 0 v w	mod 0 0 0 r/m		
ROR = Rotate Right	1 1 0 1 0 0 v w	mod 0 0 1 r/m		
RCL = Rotate Through Carry Flag Left	1 1 0 1 0 0 v w	mod 0 1 0 r/m		
RCR = Rotate Through Carry Right	1 1 0 1 0 0 v w	mod 0 1 1 r/m		
AND = And :				
Reg./Memory and Register to Either	0 0 1 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w = 1
Immediate to Accumulator	0 0 1 0 0 1 0 w	data	data if w = 1	
TEST = And Function to Flags, No Result :				
Register/Memory and Register	1 0 0 0 0 1 0 w	mod reg r/m		
Immediate Data and Register/Memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate Data and Accumulator	1 0 1 0 1 0 0 w	data	data if w = 1	
OR = Or :				
Reg./Memory and Register to Either	0 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1
Immediate to Accumulator	0 0 0 0 1 1 0 w	data	data if w = 1	
XOR = Exclusive OR :				
Reg./Memory and Register to Either	0 0 1 1 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1
Immediate to Accumulator	0 0 1 1 0 1 0 w	data	data if w = 1	
STRING MANIPULATION				
REP = Repeat	1 1 1 1 0 0 1 z			
MOVS = Move Byte/Word	1 0 1 0 0 1 0 w			
CMPS = Compare Byte/Word	1 0 1 0 0 1 1 w			
SCAS = Scan Byte/Word	1 0 1 0 1 1 1 w			
LODS = Load Byte/Wd to AL/AX	1 0 1 0 1 1 0 w			
STOS = Stor Byte/Wd from AL/A	1 0 1 0 1 0 1 w			
CONTROL TRANSFER				
CALL = Call :				
Direct Within Segment	1 1 1 0 1 0 0 0	disp-low	disp-high	
Indirect Within Segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m		
Direct Intersegment	1 0 0 1 1 0 1 0	offset-low	offset-high	
		seg-low	seg-high	
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m		

Table 2. Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code		
CONTROL TRANSFER (Continued)			
JMP = Unconditional Jump:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Direct Within Segment	1 1 1 0 1 0 0 1	disp-low	disp-high
Direct Within Segment-Short	1 1 1 0 1 0 1 1	disp	
Indirect Within Segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	
Direct Intersegment	1 1 1 0 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	
RET = Return from CALL:			
Within Segment	1 1 0 0 0 0 1 1		
Within Seg. Adding Immed to SP	1 1 0 0 0 0 1 0	data-low	data-high
Intersegment	1 1 0 0 1 0 1 1		
Intersegment Adding Immediate to SP	1 1 0 0 1 0 1 0	data-low	data-high
JE/JZ = Jump on Equal/Zero	0 1 1 1 0 1 0 0	disp	
JL/JNGE = Jump on Less/Not Greater or Equal	0 1 1 1 1 1 0 0	disp	
JLE/JNG = Jump on Less or Equal/Not Greater	0 1 1 1 1 1 1 0	disp	
JB/JNAE = Jump on Below/Not Above or Equal	0 1 1 1 0 0 1 0	disp	
JBE/JNA = Jump on Below or Equal/Not Above	0 1 1 1 0 1 1 0	disp	
JP/JPE = Jump on Parity/Parity Even	0 1 1 1 1 0 1 0	disp	
JO = Jump on Overflow	0 1 1 1 0 0 0 0	disp	
JS = Jump on Sign	0 1 1 1 1 0 0 0	disp	
JNE/JNZ = Jump on Not Equal/Not Zero	0 1 1 1 0 1 0 1	disp	
JNL/JGE = Jump on Not Less/Greater or Equal	0 1 1 1 1 1 0 1	disp	
JNLE/JG = Jump on Not Less or Equal/Greater	0 1 1 1 1 1 1 1	disp	
JNB/JAE = Jump on Not Below/Above or Equal	0 1 1 1 0 0 1 1	disp	
JNBE/JA = Jump on Not Below or Equal/Above	0 1 1 1 0 1 1 1	disp	
JNP/JPO = Jump on Not Par/Par Odd	0 1 1 1 1 0 1 1	disp	
JNO = Jump on Not Overflow	0 1 1 1 0 0 0 1	disp	
JNS = Jump on Not Sign	0 1 1 1 1 0 0 1	disp	
LOOP = Loop CX Times	1 1 1 0 0 0 1 0	disp	
LOOPZ/LOOPE = Loop While Zero/Equal	1 1 1 0 0 0 0 1	disp	
LOOPNZ/LOOPNE = Loop While Not Zero/Equal	1 1 1 0 0 0 0 0	disp	
JCZX = Jump on CX Zero	1 1 1 0 0 0 1 1	disp	
INT = Interrupt			
Type Specified	1 1 0 0 1 1 0 1	type	
Type 3	1 1 0 0 1 1 0 0		
INTO = Interrupt on Overflow	1 1 0 0 1 1 1 0		
IRET = Interrupt Return	1 1 0 0 1 1 1 1		

Table 2. Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code	
PROCESSOR CONTROL	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
CLC = Clear Carry	1 1 1 1 1 0 0 0	
CMC = Complement Carry	1 1 1 1 0 1 0 1	
STC = Set Carry	1 1 1 1 1 0 0 1	
CLD = Clear Direction	1 1 1 1 1 1 0 0	
STD = Set Direction	1 1 1 1 1 1 0 1	
CLI = Clear Interrupt	1 1 1 1 1 0 1 0	
STI = Set Interrupt	1 1 1 1 1 0 1 1	
HLT = Halt	1 1 1 1 0 1 0 0	
WAIT = Wait	1 0 0 1 1 0 1 1	
ESC = Escape (to External Device)	1 1 0 1 1 x x x	mod x x x r/m
LOCK = Bus Lock Prefix	1 1 1 1 0 0 0 0	

NOTES:

AL = 8-bit accumulator
 AX = 16-bit accumulator
 CX = Count register
 DS = Data segment
 ES = Extra segment
 Above/below refers to unsigned value.
 Greater = more positive:
 Less = less positive (more negative) signed values
 if d = 1 then "to" reg; if d = 0 then "from" reg
 if w = 1 then word instruction; if w = 0 then byte instruction
 if mod = 11 then r/m is treated as a REG field
 if mod = 00 then DISP = 0*, disp-low and disp-high are absent
 if mod = 01 then DISP = disp-low sign-extended to 16 bits, disp-high is absent
 if mod = 10 then DISP = disp-high: disp-low
 if r/m = 000 then EA = (BX) + (SI) + DISP
 if r/m = 001 then EA = (BX) + (DI) + DISP
 if r/m = 010 then EA = (BP) + (SI) + DISP
 if r/m = 011 then EA = (BP) + (DI) + DISP
 if r/m = 100 then EA = (SI) + DISP
 if r/m = 101 then EA = (DI) + DISP
 if r/m = 110 then EA = (BP) + DISP*
 if r/m = 111 then EA = (BX) + DISP
 DISP follows 2nd byte of instruction (before data if required)
 *except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.
 **MOV CS, REG/MEMORY not allowed.

if s w = 01 then 16 bits of immediate data form the operand
 if s w = 11 then an immediate data byte is sign extended to form the 16-bit operand
 if v = 0 then "count" = 1; if v = 1 then "count" in (CL) register
 x = don't care
 z is used for string primitives for comparison with ZF FLAG

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:

FLAGS =
 X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

Mnemonics © Intel, 1978

DATA SHEET REVISION REVIEW

The following list represents key differences between this and the -001 data sheet. Please review this summary carefully.

- In the Pin Description Table (Table 1), the description of the HLDA signal being issued has been corrected. HLDA will be issued in the middle of either the T₄ or T₁ state.



8088

8-BIT HMOS MICROPROCESSOR

8088/8088-2

- 8-Bit Data Bus Interface
- 16-Bit Internal Architecture
- Direct Addressing Capability to 1 Mbyte of Memory
- Direct Software Compatibility with 8086 CPU
- 14-Word by 16-Bit Register Set with Symmetrical Operations
- 24 Operand Addressing Modes
- Byte, Word, and Block Operations
- 8-Bit and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal, Including Multiply and Divide
- Two Clock Rates:
 - 5 MHz for 8088
 - 8 MHz for 8088-2
- Available in EXPRESS
 - Standard Temperature Range
 - Extended Temperature Range

The Intel® 8088 is a high performance microprocessor implemented in N-channel, depletion load, silicon gate technology (HMOS), and packaged in a 40-pin CERDIP package. The processor has attributes of both 8- and 16-bit microprocessors. It is directly compatible with 8086 software and 8080/8085 hardware and peripherals.

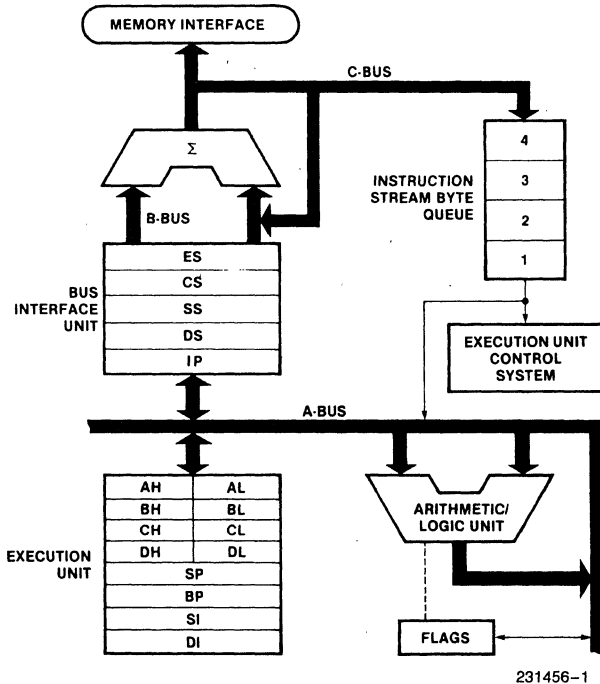


Figure 1. 8088 CPU Functional Block Diagram

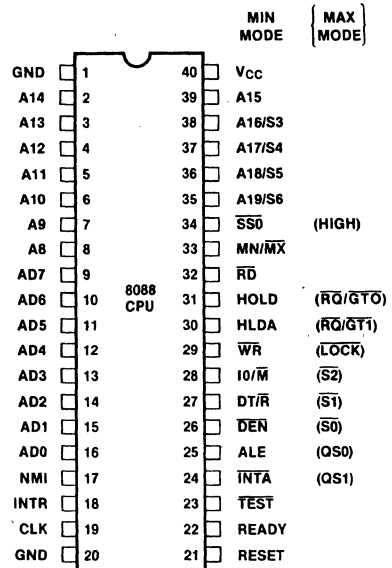


Figure 2. 8088 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for 8088 systems in either minimum or maximum mode. The "local bus" in these descriptions is the direct multiplexed bus interface connection to the 8088 (without regard to additional bus buffers).

Symbol	Pin No.	Type	Name and Function															
AD7-AD0	9-16	I/O	ADDRESS DATA BUS: These lines constitute the time multiplexed memory/I/O address (T1) and data (T2, T3, Tw, T4) bus. These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".															
A15-A8	2-8, 39	O	ADDRESS BUS: These lines provide address bits 8 through 15 for the entire bus cycle (T1-T4). These lines do not have to be latched by ALE to remain valid. A15-A8 are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".															
A19/S6, A18/S5, A17/S4, A16/S3	35-38	O	<p>ADDRESS/STATUS: During T1, these are the four most significant address lines for memory operations. During I/O operations, these lines are LOW. During memory and I/O operations, status information is available on these lines during T2, T3, Tw, and T4. S6 is always low. The status of the interrupt enable flag bit (S5) is updated at the beginning of each clock cycle. S4 and S3 are encoded as shown. This information indicates which segment register is presently being used for data accessing. These lines float to 3-state OFF during local bus "hold acknowledge".</p> <table border="1"> <thead> <tr> <th>S4</th> <th>S3</th> <th>Characteristics</th> </tr> </thead> <tbody> <tr> <td>0 (LOW)</td> <td>0</td> <td>Alternate Data</td> </tr> <tr> <td>0</td> <td>1</td> <td>Stack</td> </tr> <tr> <td>1 (HIGH)</td> <td>0</td> <td>Code or None</td> </tr> <tr> <td>1</td> <td>1</td> <td>Data</td> </tr> </tbody> </table> <p>S6 is 0 (LOW)</p>	S4	S3	Characteristics	0 (LOW)	0	Alternate Data	0	1	Stack	1 (HIGH)	0	Code or None	1	1	Data
S4	S3	Characteristics																
0 (LOW)	0	Alternate Data																
0	1	Stack																
1 (HIGH)	0	Code or None																
1	1	Data																
RD	32	O	READ: Read strobe indicates that the processor is performing a memory or I/O read cycle, depending on the state of the IO/M pin or S2. This signal is used to read devices which reside on the 8088 local bus. RD is active LOW during T2, T3 and Tw of any read cycle, and is guaranteed to remain HIGH in T2 until the 8088 local bus has floated. This signal floats to 3-state OFF in "hold acknowledge".															
READY	22	I	READY: is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The RDY signal from memory or I/O is synchronized by the 8284 clock generator to form READY. This signal is active HIGH. The 8088 READY input is not synchronized. Correct operation is not guaranteed if the set up and hold times are not met.															
INTR	18	I	INTERRUPT REQUEST: is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.															
TEST	23	I	TEST: input is examined by the "wait for test" instruction. If the TEST input is LOW, execution continues, otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.															

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
NMI	17	I	NON-MASKABLE INTERRUPT: is an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from a LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.
RESET	21	I	RESET: causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the instruction set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	CLOCK: provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V _{CC}	40		V_{CC}: is the +5V ± 10% power supply pin.
GND	1, 20		GND: are the ground pins.
MN/ $\overline{M\bar{X}}$	33	I	MINIMUM/MAXIMUM: indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

The following pin function descriptions are for the 8088 minimum mode (i.e., $MN/\overline{M\bar{X}} = V_{CC}$). Only the pin functions which are unique to minimum mode are described; all other pin functions are as described above.

Symbol	Pin No.	Type	Name and Function
IO/\overline{M}	28	O	STATUS LINE: is an inverted maximum mode $\overline{S2}$. It is used to distinguish a memory access from an I/O access. IO/\overline{M} becomes valid in the T4 preceding a bus cycle and remains valid until the final T4 of the cycle ($I/O = \text{HIGH}$, $M = \text{LOW}$). IO/\overline{M} floats to 3-state OFF in local bus "hold acknowledge".
\overline{WR}	29	O	WRITE: strobe indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the IO/\overline{M} signal. \overline{WR} is active for T2, T3, and Tw of any write cycle. It is active LOW, and floats to 3-state OFF in local bus "hold acknowledge".
\overline{INTA}	24	O	INTA: is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T2, T3, and Tw of each interrupt acknowledge cycle.
ALE	25	O	ADDRESS LATCH ENABLE: is provided by the processor to latch the address into an address latch. It is a HIGH pulse active during clock low of T1 of any bus cycle. Note that ALE is never floated.
DT/\overline{R}	27	O	DATA TRANSMIT/RECEIVE: is needed in a minimum system that desires to use a data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically, DT/\overline{R} is equivalent to $\overline{S1}$ in the maximum mode, and its timing is the same as for IO/\overline{M} ($T = \text{HIGH}$, $R = \text{LOW}$). This signal floats to 3-state OFF in local "hold acknowledge".
\overline{DEN}	26	O	DATA ENABLE: is provided as an output enable for the data bus transceiver in a minimum system which uses the transceiver. \overline{DEN} is active LOW during each memory and I/O access, and for \overline{INTA} cycles. For a read or \overline{INTA} cycle, it is active from the middle of T2 until the middle of T4, while for a write cycle, it is active from the beginning of T2 until the middle of T4. \overline{DEN} floats to 3-state OFF during local bus "hold acknowledge".

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function			
HOLD, HLDA	31, 30	I, O	<p>HOLD: indicates that another master is requesting a local bus "hold". To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement, in the middle of a T4 or T_i clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor lowers HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. HOLD and HLDA have internal pull-up resistors.</p> <p>Hold is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the set up time.</p>			
\overline{SSO}	34	O	<p>STATUS LINE: is logically equivalent to $\overline{S0}$ in the maximum mode. The combination of \overline{SSO}, IO/\overline{M} and DT/\overline{R} allows the system to completely decode the current bus cycle status.</p>			
			IO/\overline{M}	DT/\overline{R}	\overline{SSO}	Characteristics
			1(HIGH)	0	0	Interrupt Acknowledge
			1	0	1	Read I/O Port
			1	1	0	Write I/O Port
			1	1	1	Halt
			0(LOW)	0	0	Code Access
			0	0	1	Read Memory
			0	1	0	Write Memory
			0	1	1	Passive

The following pin function descriptions are for the 8088/8288 system in maximum mode (i.e., $MN/\overline{MX} = GND$). Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

Symbol	Pin No.	Type	Name and Function			
$\overline{S2}, \overline{S1}, \overline{S0}$	26-28	O	<p>STATUS: is active during clock high of T4, T1, and T2, and is returned to the passive state (1,1,1) during T3 or during Tw when READY is HIGH. This status is used by the 8288 bus controller to generate all memory and I/O access control signals. Any change by $\overline{S2}$, $\overline{S1}$, or $\overline{S0}$ during T4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T3 and Tw is used to indicate the end of a bus cycle.</p> <p>These signals float to 3-state OFF during "hold acknowledge". During the first clock cycle after RESET becomes active, these signals are active HIGH. After this first clock, they float to 3-state OFF.</p>			
			$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Characteristics
			0(LOW)	0	0	Interrupt Acknowledge
			0	0	1	Read I/O Port
			0	1	0	Write I/O Port
			0	1	1	Halt
			1(HIGH)	0	0	Code Access
			1	0	1	Read Memory
			1	1	0	Write Memory
			1	1	1	Passive

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function															
$\overline{RQ}/\overline{GT0}$, $RQ/\overline{GT1}$	30, 31	I/O	<p>REQUEST/GRANT: pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT0}$ having higher priority than $RQ/\overline{GT1}$. RQ/\overline{GT} has an internal pull-up resistor, so may be left unconnected. The request/grant sequence is as follows (See Figure 8):</p> <ol style="list-style-type: none"> 1. A pulse of one CLK wide from another local bus master indicates a local bus request ("hold") to the 8088 (pulse 1). 2. During a T4 or T1 clock cycle, a pulse one clock wide from the 8088 to the requesting master (pulse 2), indicates that the 8088 has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge". The same rules as for HOLD/HOLDA apply as for when the bus is released. 3. A pulse one CLK wide from the requesting master indicates to the 8088 (pulse 3) that the "hold" request is about to end and that the 8088 can reclaim the local bus at the next CLK. The CPU then enters T4. <p>Each master-master exchange of the local bus is a sequence of three pulses. There must be one idle CLK cycle after each bus exchange. Pulses are active LOW.</p> <p>If the request is made while the CPU is performing a memory cycle, it will release the local bus during T4 of the cycle when all the following conditions are met:</p> <ol style="list-style-type: none"> 1. Request occurs on or before T2. 2. Current cycle is not the low bit of a word. 3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence. 4. A locked instruction is not currently executing. <p>If the local bus is idle when the request is made the two possible events will follow:</p> <ol style="list-style-type: none"> 1. Local bus will be released during the next clock. 2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied. 															
LOCK	29	O	<p>LOCK: indicates that other system bus masters are not to gain control of the system bus while LOCK is active (LOW). The LOCK signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state off in "hold acknowledge".</p>															
QS1, QS0	24, 25	O	<p>QUEUE STATUS: provide status to allow external tracking of the internal 8088 instruction queue. The queue status is valid during the CLK cycle after which the queue operation is performed.</p> <table border="1"> <thead> <tr> <th>QS1</th> <th>QS0</th> <th>Characteristics</th> </tr> </thead> <tbody> <tr> <td>0(LOW)</td> <td>0</td> <td>No Operation</td> </tr> <tr> <td>0</td> <td>1</td> <td>First Byte of Opcode from Queue</td> </tr> <tr> <td>1(HIGH)</td> <td>0</td> <td>Empty the Queue</td> </tr> <tr> <td>1</td> <td>1</td> <td>Subsequent Byte from Queue</td> </tr> </tbody> </table>	QS1	QS0	Characteristics	0(LOW)	0	No Operation	0	1	First Byte of Opcode from Queue	1(HIGH)	0	Empty the Queue	1	1	Subsequent Byte from Queue
QS1	QS0	Characteristics																
0(LOW)	0	No Operation																
0	1	First Byte of Opcode from Queue																
1(HIGH)	0	Empty the Queue																
1	1	Subsequent Byte from Queue																
—	34	O	Pin 34 is always high in the maximum mode.															

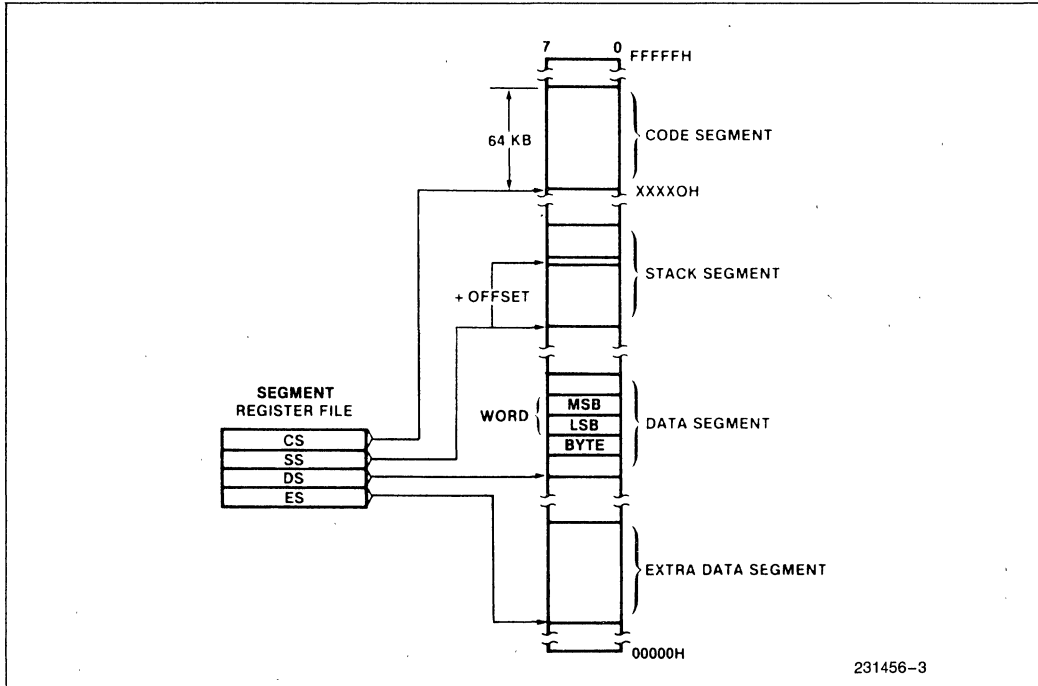


Figure 3. Memory Organization

FUNCTIONAL DESCRIPTION

Memory Organization

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64K bytes each, with each segment falling on 16-byte boundaries (See Figure 3).

All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the ad-

ressing needs of programs. The segment register to be selected is automatically chosen according to the rules of the following table. All information in one segment type share the same logical attributes (e.g. code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster, and more structured.

Word (16-bit) operands can be located on even or odd address boundaries. For address and data operands, the least significant byte of the word is stored in the lower valued address location and the most significant byte in the next higher address location. The BIU will automatically execute two fetch or write cycles for 16-bit operands.

Memory Reference Used	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch.
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
Local Data	DATA (DS)	Data references when: relative to stack, destination of string operation, or explicitly overridden.
External (Global) Data	EXTRA (ES)	Destination of string operations: Explicitly selected using a segment override.

Certain locations in memory are reserved for specific CPU operations (See Figure 4). Locations from addresses FFFF0H through FFFFFH are reserved for operations including a jump to the initial system initialization routine. Following RESET, the CPU will always begin execution at location FFFF0H where the jump must be located. Locations 00000H through 003FFH are reserved for interrupt operations. Four-byte pointers consisting of a 16-bit segment address and a 16-bit offset address direct program flow to one of the 256 possible interrupt service routines. The pointer elements are assumed to have been stored at their respective places in reserved memory prior to the occurrence of interrupts.

Minimum and Maximum Modes

The requirements for supporting minimum and maximum 8088 systems are sufficiently different that they cannot be done efficiently with 40 uniquely defined pins. Consequently, the 8088 is equipped with a strap pin (MN/MX) which defines the system con-

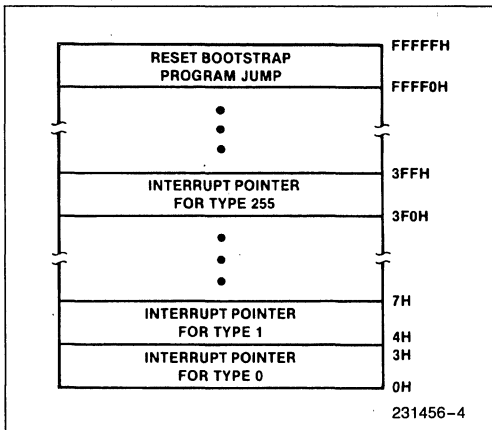


Figure 4. Reserved Memory Locations

figuration. The definition of a certain subset of the pins changes, dependent on the condition of the strap pin. When the MN/MX pin is strapped to GND, the 8088 defines pins 24 through 31 and 34 in maximum mode. When the MN/MX pin is strapped to V_{CC} , the 8088 generates bus control signals itself on pins 24 through 31 and 34.

The minimum mode 8088 can be used with either a multiplexed or demultiplexed bus. The multiplexed bus configuration is compatible with the MCS-85™ multiplexed bus peripherals. This configuration (See Figure 5) provides the user with a minimum chip count system. This architecture provides the 8088 processing power in a highly integrated form.

The demultiplexed mode requires one latch (for 64K addressability) or two latches (for a full megabyte of addressing). A third latch can be used for buffering if the address bus loading requires it. A transceiver can also be used if data bus buffering is required (See Figure 6). The 8088 provides \overline{DEN} and DT/\overline{R} to control the transceiver, and ALE to latch the addresses. This configuration of the minimum mode provides the standard demultiplexed bus structure with heavy bus buffering and relaxed bus timing requirements.

The maximum mode employs the 8288 bus controller (See Figure 7). The 8288 decodes status lines S_0 , S_1 , and S_2 , and provides the system with all bus control signals. Moving the bus control to the 8288 provides better source and sink current capability to the control lines, and frees the 8088 pins for extended large system features. Hardware lock, queue status, and two request/grant interfaces are provided by the 8088 in maximum mode. These features allow co-processors in local bus and remote bus configurations.

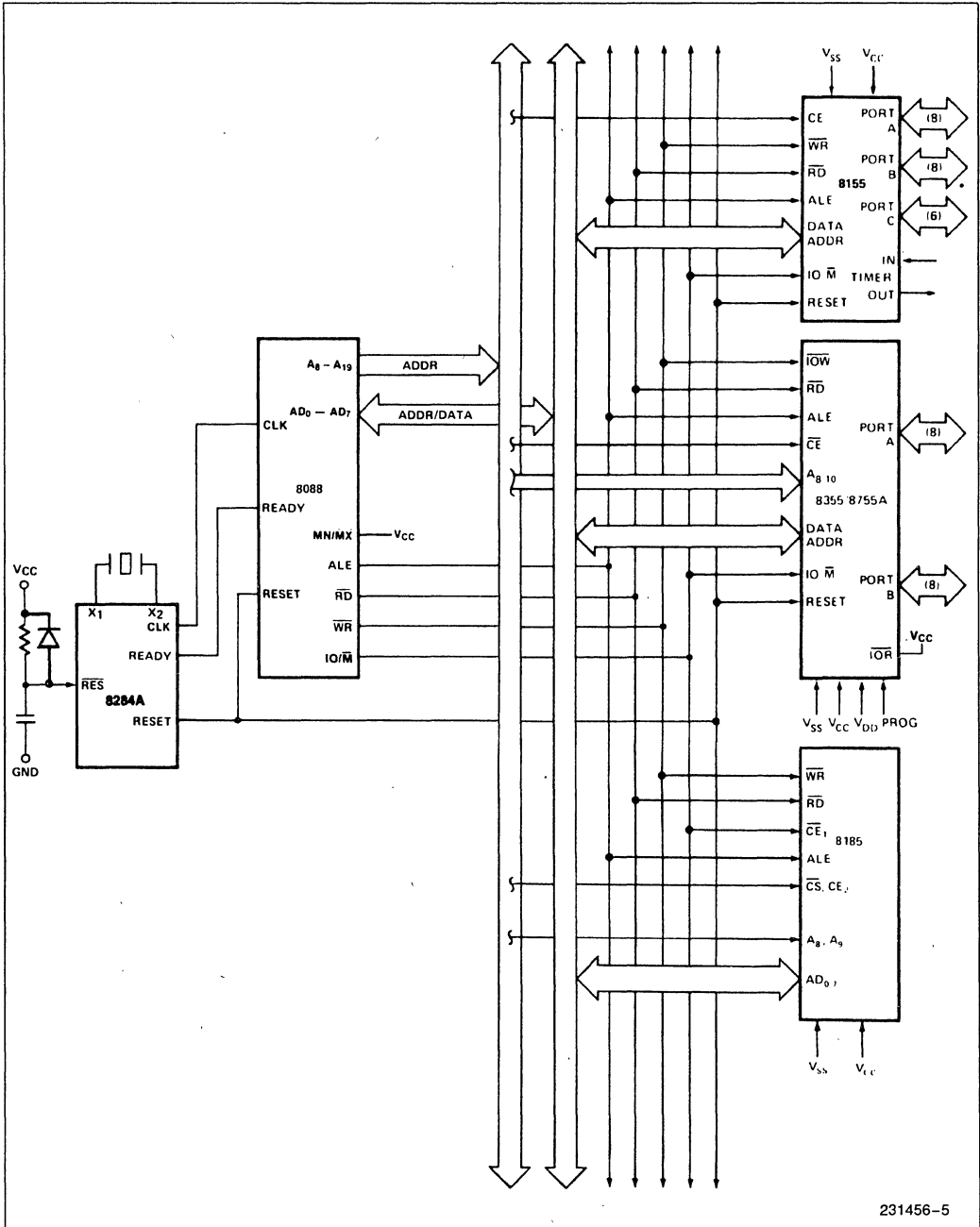


Figure 5. Multiplexed Bus Configuration

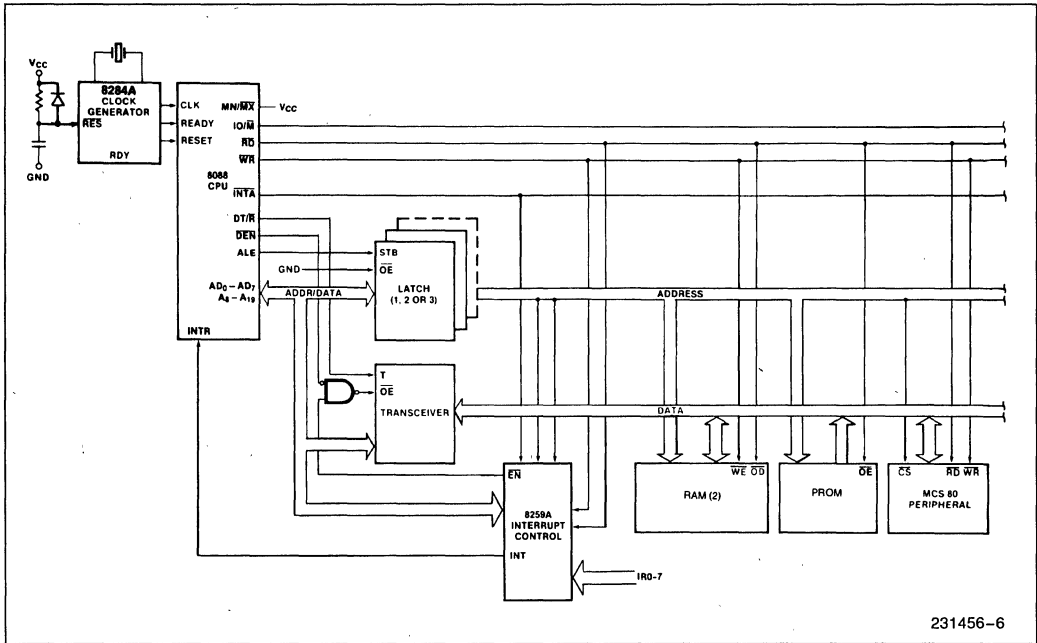


Figure 6. Demultiplexed Bus Configuration

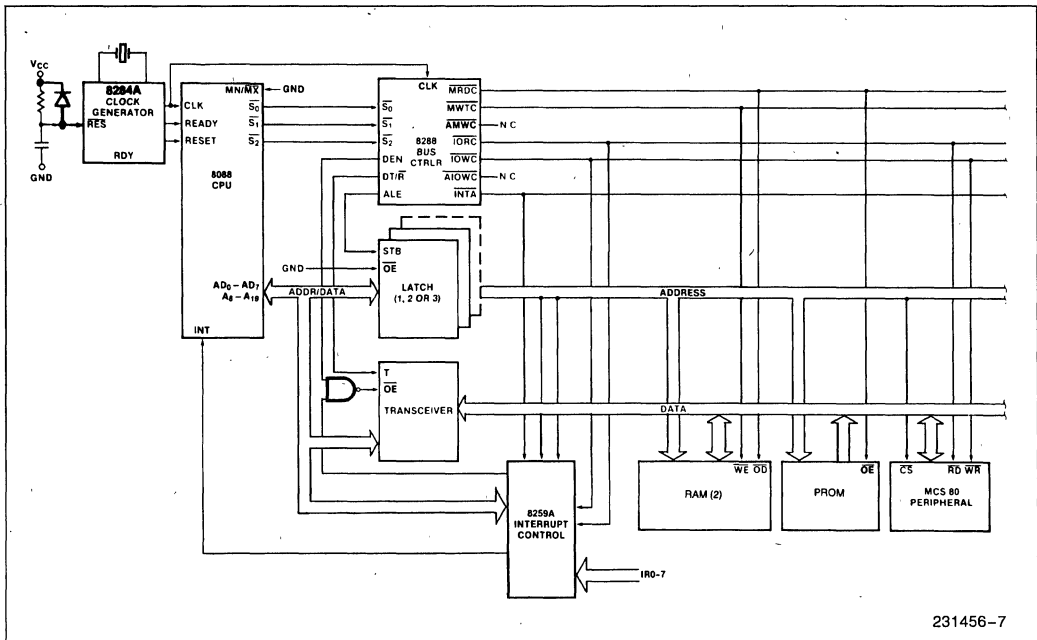


Figure 7. Fully Buffered System Using Bus Controller

Bus Operation

The 8088 address/data bus is broken into three parts—the lower eight address/data bits (AD0-AD7), the middle eight address bits (A8-A15), and the upper four address bits (A16-A19). The address/data bits and the highest four address bits are time multiplexed. This technique provides the most efficient use of pins on the processor, permitting the use of a standard 40 lead package. The middle eight address bits are not multiplexed, i.e. they remain val-

id throughout each bus cycle. In addition, the bus can be demultiplexed at the processor with a single address latch if a standard, non-multiplexed bus is desired for the system.

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T1, T2, T3, and T4 (See Figure 8). The address is emitted from the processor during T1 and data transfer occurs on the bus during T3 and T4. T2 is used primarily for chang-

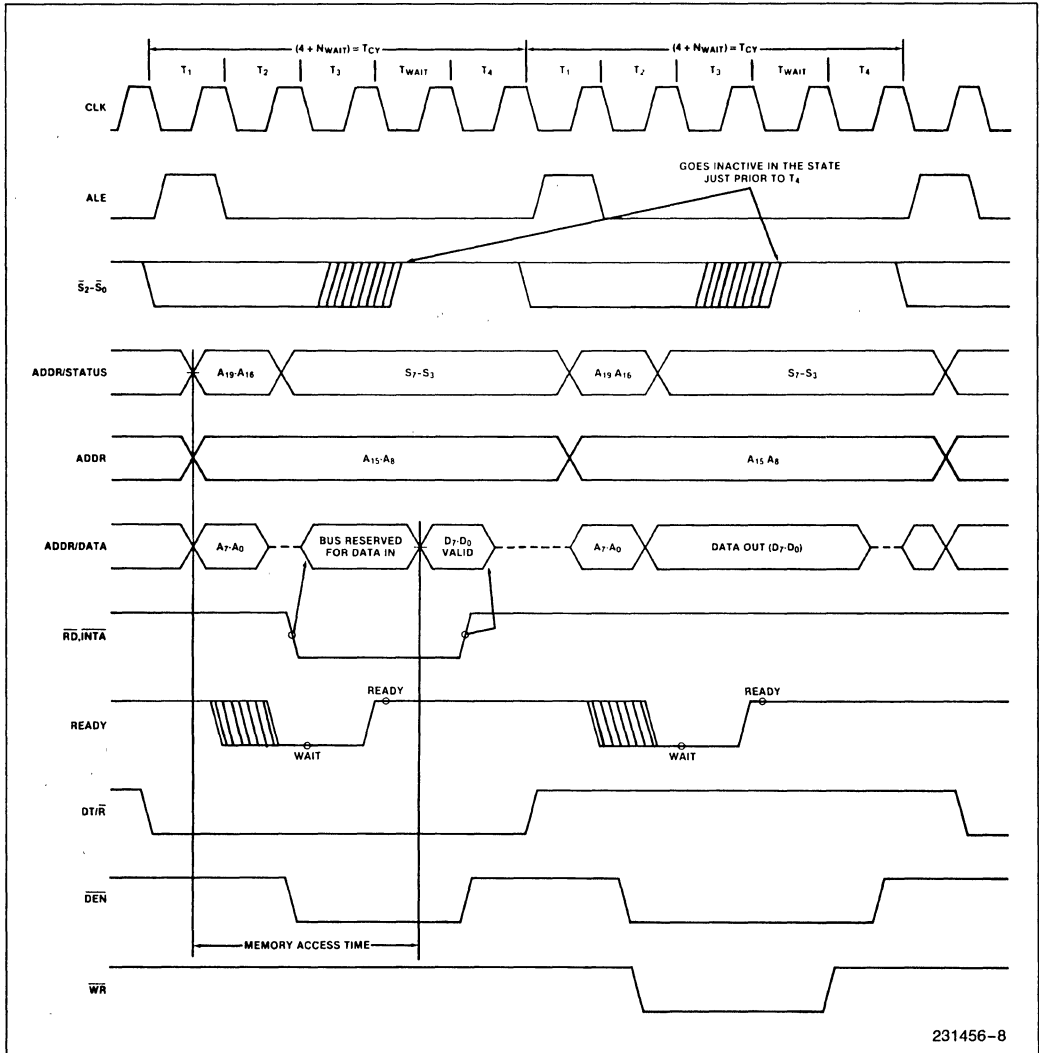


Figure 8. Basic System Timing

231456-8

ing the direction of the bus during read operations. In the event that a "NOT READY" indication is given by the addressed device, "wait" states (T_w) are inserted between T_3 and T_4 . Each inserted "wait" state is of the same duration as a CLK cycle. Periods can occur between 8088 driven bus cycles. These are referred to as "idle" states (T_i), or inactive CLK cycles. The processor uses these cycles for internal housekeeping.

During T_1 of any bus cycle, the ALE (address latch enable) signal is emitted (by either the processor or the 8288 bus controller, depending on the MN/MX strap). At the trailing edge of this pulse, a valid address and certain status information for the cycle may be latched.

Status bits $\overline{S_0}$, $\overline{S_1}$, and $\overline{S_2}$ are used by the bus controller, in maximum mode, to identify the type of bus transaction according to the following table:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Characteristics
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (No Bus Cycle)

Status bits S_3 through S_6 are multiplexed with high order address bits and are therefore valid during T_2 through T_4 . S_3 and S_4 indicate which segment register was used for this bus cycle in forming the address according to the following table:

S_4	S_3	Characteristics
0 (LOW)	0	Alternate Data (Extra Segment)
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data

S_5 is a reflection of the PSW interrupt enable bit. S_6 is always equal to 0.

I/O Addressing

In the 8088, I/O operations can address up to a maximum of 64K I/O registers. The I/O address appears in the same format as the memory address on bus lines A_{15} – A_0 . The address lines A_{19} – A_{16} are zero in I/O operations. The variable I/O instructions,

which use register DX as a pointer, have full address capability, while the direct I/O instructions directly address one or two of the 256 I/O byte locations in page 0 of the I/O address space. I/O ports are addressed in the same manner as memory locations.

Designers familiar with the 8085 or upgrading an 8085 design should note that the 8085 addresses I/O with an 8-bit address on both halves of the 16-bit address bus. The 8088 uses a full 16-bit address on its lower 16 address lines.

EXTERNAL INTERFACE

Processor Reset and Initialization

Processor initialization or start up is accomplished with activation (HIGH) of the RESET pin. The 8088 RESET is required to be HIGH for greater than four clock cycles. The 8088 will terminate operations on the high-going edge of RESET and will remain dormant as long as RESET is HIGH. The low-going transition of RESET triggers an internal reset sequence for approximately 7 clock cycles. After this interval the 8088 operates normally, beginning with the instruction in absolute locations FFFF0H (See Figure 4). The RESET input is internally synchronized to the processor clock. At initialization, the HIGH to LOW transition of RESET must occur no sooner than 50 μ s after power up, to allow complete initialization of the 8088.

NMI asserted prior to the 2nd clock after the end of RESET will not be honored. If NMI is asserted after that point and during the internal reset sequence, the processor may execute one instruction before responding to the interrupt. A hold request active immediately after RESET will be honored before the first instruction fetch.

All 3-state outputs float to 3-state OFF during RESET. Status is active in the idle state for the first clock after RESET becomes active and then floats to 3-state OFF. ALE and HLDA are driven low.

Interrupt Operations

Interrupt operations fall into two classes: software or hardware initiated. The software initiated interrupts and software aspects of hardware interrupts are specified in the instruction set description in the iAPX 88 book or the iAPX 86,88 User's Manual. Hardware interrupts can be classified as nonmaskable or maskable.

Interrupts result in a transfer of control to a new program location. A 256 element table containing address pointers to the interrupt service program locations resides in absolute locations 0 through 3FFH (See Figure 4), which are reserved for this purpose. Each element in the table is 4 bytes in size and corresponds to an interrupt "type." An interrupting device supplies an 8-bit type number, during the interrupt acknowledge sequence, which is used to vector through the appropriate element to the new interrupt service program location.

Non-Maskable Interrupt (NMI)

The processor provides a single non-maskable interrupt (NMI) pin which has higher priority than the maskable interrupt request (INTR) pin. A typical use would be to activate a power failure routine. The NMI is edge-triggered on a LOW to HIGH transition. The activation of this pin causes a type 2 interrupt.

NMI is required to have a duration in the HIGH state of greater than two clock cycles, but is not required to be synchronized to the clock. Any higher going transition of NMI is latched on-chip and will be serviced at the end of the current instruction or between whole moves (2 bytes in the case of word moves) of a block type instruction. Worst case response to NMI would be for multiply, divide, and variable shift instructions. There is no specification on the occurrence of the low-going edge; it may occur before, during, or after the servicing of NMI. Another high-going edge triggers another response if it occurs after the start of the NMI procedure. The signal must be free of logical spikes in general and be free of bounces on the low-going edge to avoid triggering extraneous responses.

Maskable Interrupt (INTR)

The 8088 provides a single interrupt request input (INTR) which can be masked internally by software with the resetting of the interrupt enable (IF) flag bit. The interrupt request signal is level triggered. It is internally synchronized during each clock cycle on the high-going edge of CLK. To be responded to, INTR must be present (HIGH) during the clock period preceding the end of the current instruction or the end of a whole move for a block type instruction. During interrupt response sequence, further interrupts are disabled. The enable bit is reset as part of the response to any interrupt (INTR, NMI, software interrupt, or single step), although the FLAGS register which is automatically pushed onto the stack reflects the state of the processor prior to the interrupt. Until the old FLAGS register is restored, the

enable bit will be zero unless specifically set by an instruction.

During the response sequence (See Figure 9), the processor executes two successive (back to back) interrupt acknowledge cycles. The 8088 emits the LOCK signal (maximum mode only) from T2 of the first bus cycle until T2 of the second. A local bus "hold" request will not be honored until the end of the second bus cycle. In the second bus cycle, a byte is fetched from the external interrupt system (e.g., 8259A PIC) which identifies the source (type) of the interrupt. This byte is multiplied by four and used as a pointer into the interrupt vector lookup table. An INTR signal left HIGH will be continually responded to within the limitations of the enable bit and sample period. The interrupt return instruction includes a flags pop which returns the status of the original interrupt enable bit when it restores the flags.

HALT

When a software HALT instruction is executed, the processor indicates that it is entering the HALT state in one of two ways, depending upon which mode is strapped. In minimum mode, the processor issues ALE, delayed by one clock cycle, to allow the system to latch the halt status. Halt status is available on $\text{IO}/\overline{\text{M}}$, $\text{DT}/\overline{\text{R}}$, and $\overline{\text{SSO}}$. In maximum mode, the processor issues appropriate HALT status on $\overline{\text{S2}}$, $\overline{\text{S1}}$, and $\overline{\text{S0}}$, and the 8288 bus controller issues one ALE. The 8088 will not leave the HALT state when a local bus hold is entered while in HALT. In this case, the processor reissues the HALT indicator at the end of the local bus hold. An interrupt request or RESET will force the 8088 out of the HALT state.

Read/Modify/Write (Semaphore) Operations via LOCK

The LOCK status information is provided by the processor when consecutive bus cycles are required during the execution of an instruction. This allows the processor to perform read/modify/write operations on memory (via the "exchange register with memory" instruction), without another system bus master receiving intervening memory cycles. This is useful in multiprocessor system configurations to accomplish "test and set lock" operations. The $\overline{\text{LOCK}}$ signal is activated (LOW) in the clock cycle following decoding of the LOCK prefix instruction. It is deactivated at the end of the last bus cycle of the instruction following the LOCK prefix. While $\overline{\text{LOCK}}$ is active, a request on a $\overline{\text{RQ}}/\overline{\text{GT}}$ pin will be recorded, and then honored at the end of the LOCK.

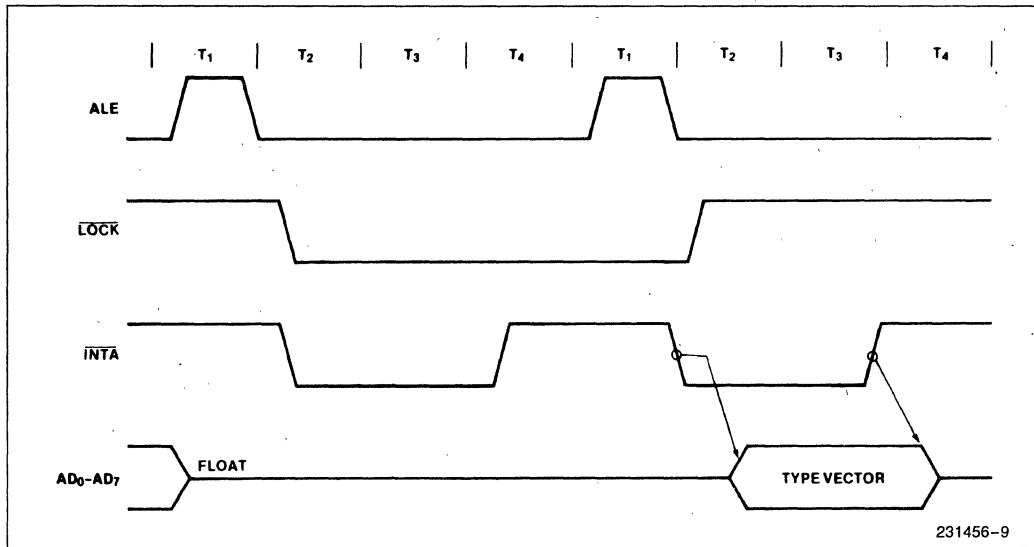


Figure 9. Interrupt Acknowledge Sequence

External Synchronization via $\overline{\text{TEST}}$

As an alternative to interrupts, the 8088 provides a single software-testable input pin ($\overline{\text{TEST}}$). This input is utilized by executing a WAIT instruction. The single WAIT instruction is repeatedly executed until the $\overline{\text{TEST}}$ input goes active (LOW). The execution of WAIT does not consume bus cycles once the queue is full.

If a local bus request occurs during WAIT execution, the 8088 3-states all output drivers. If interrupts are enabled, the 8088 will recognize interrupts and process them. The WAIT instruction is then refetched, and reexecuted.

Basic System Timing

In minimum mode, the $\text{MN}/\overline{\text{MX}}$ pin is strapped to V_{CC} and the processor emits bus control signals compatible with the 8085 bus structure. In maximum mode, the $\text{MN}/\overline{\text{MX}}$ pin is strapped to GND and the processor emits coded status information which the 8288 bus controller uses to generate MULTIBUS compatible bus control signals.

System Timing—Minimum System

(See Figure 8)

The read cycle begins in T1 with the assertion of the address latch enable (ALE) signal. The trailing (low

going) edge of this signal is used to latch the address information, which is valid on the address/data bus (AD0-AD7) at this time, into the 8282/8283 latch. Address lines A8 through A15 do not need to be latched because they remain valid throughout the bus cycle. From T1 to T4 the $\text{IO}/\overline{\text{M}}$ signal indicates a memory or I/O operation. At T2 the address is removed from the address/data bus and the bus goes to a high impedance state. The read control signal is also asserted at T2. The read ($\overline{\text{RD}}$) signal causes the addressed device to enable its data bus drivers to the local bus. Some time later, valid data will be available on the bus and the addressed device will drive the READY line HIGH. When the processor returns the read signal to a HIGH level, the addressed device will again 3-state its bus drivers. If a transceiver is required to buffer the 8088 local bus, signals DT/R and DEN are provided by the 8088.

A write cycle also begins with the assertion of ALE and the emission of the address. The $\text{IO}/\overline{\text{M}}$ signal is again asserted to indicate a memory or I/O write operation. In T2, immediately following the address emission, the processor emits the data to be written into the addressed location. This data remains valid until at least the middle of T4. During T2, T3, and T4, the processor asserts the write control signal. The write ($\overline{\text{WR}}$) signal becomes active at the beginning of T2, as opposed to the read, which is delayed somewhat into T2 to provide time for the bus to float.

The basic difference between the interrupt acknowledge cycle and a read cycle is that the interrupt acknowledge ($\overline{\text{INTA}}$) signal is asserted in place of the read ($\overline{\text{RD}}$) signal and the address bus is floated. (See Figure 9) In the second of two successive $\overline{\text{INTA}}$ cycles, a byte of information is read from the data bus, as supplied by the interrupt system logic (i.e. 8259A priority interrupt controller). This byte identifies the source (type) of the interrupt. It is multiplied by four and used as a pointer into the interrupt vector lookup table, as described earlier.

Bus Timing—Medium Complexity Systems

(See Figure 10)

For medium complexity systems, the $\text{MN}/\overline{\text{MX}}$ pin is connected to GND and the 8288 bus controller is added to the system, as well as a latch for latching the system address, and a transceiver to allow for bus loading greater than the 8088 is capable of handling. Signals ALE , $\overline{\text{DEN}}$, and $\text{DT}/\overline{\text{R}}$ are generated by the 8288 instead of the processor in this configuration, although their timing remains relatively the same. The 8088 status outputs (S_2 , S_1 , and S_0) provide type of cycle information and become 8288 inputs. This bus cycle information specifies read (code, data, or I/O), write (data or I/O), interrupt acknowledge, or software halt. The 8288 thus issues control signals specifying memory read or write, I/O read or write, or interrupt acknowledge. The 8288 provides two types of write strobes, normal and advanced, to be applied as required. The normal write strobes have data valid at the leading edge of write. The advanced write strobes have the same timing as read strobes, and hence, data is not valid at the leading edge of write. The transceiver receives the usual T and $\overline{\text{OE}}$ inputs from the 8288's $\text{DT}/\overline{\text{R}}$ and DEN outputs.

The pointer into the interrupt vector table, which is passed during the second $\overline{\text{INTA}}$ cycle, can derive from an 8259A located on either the local bus or the system bus. If the master 8289A priority interrupt controller is positioned on the local bus, a TTL gate is required to disable the transceiver when reading from the master 8259A during the interrupt acknowledge sequence and software "poll".

The 8088 Compared to the 8086

The 8088 CPU is an 8-bit processor designed around the 8086 internal structure. Most internal functions of the 8088 are identical to the equivalent 8086 functions. The 8088 handles the external bus

the same way the 8086 does with the distinction of handling only 8 bits at a time. Sixteen-bit operands are fetched or written in two consecutive bus cycles. Both processors will appear identical to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions have the same end result. The differences between the 8088 and 8086 are outlined below. The engineer who is unfamiliar with the 8086 is referred to the iAPX 86, 88 User's Manual, Chapters 2 and 4, for function description and instruction set information. Internally, there are three differences between the 8088 and the 8086. All changes are related to the 8-bit bus interface.

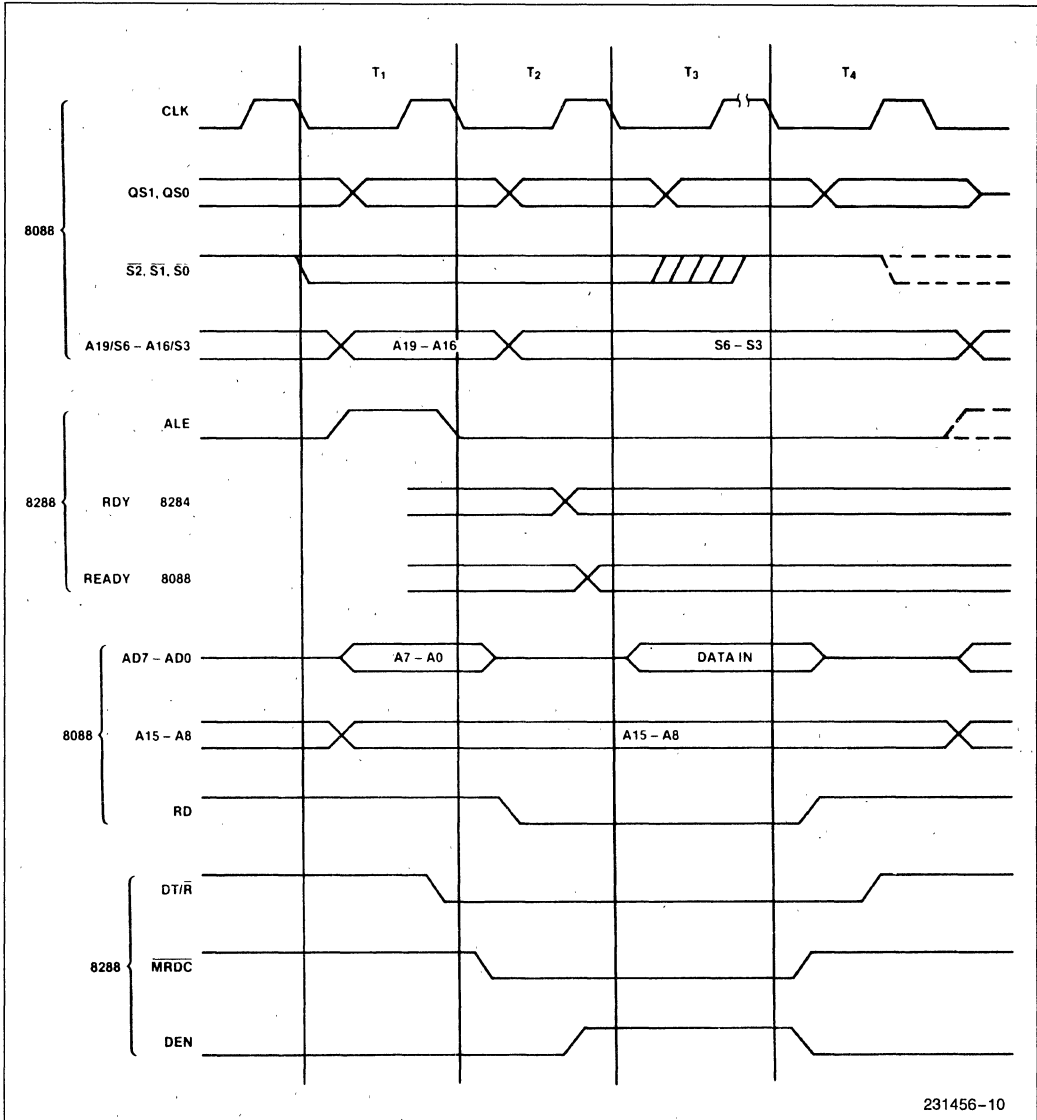
- The queue length is 4 bytes in the 8088, whereas the 8086 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The 8088 BIU will fetch a new instruction to load into the queue each time there is a 1 byte hole (space available) in the queue. The 8086 waits until a 2-byte space is available.
- The internal execution time of the instruction set is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU is also limited by the speed of instruction fetches. This latter problem only occurs when a series of simple operations occur. When the more sophisticated instructions of the 8088 are being used, the queue has time to fill and the execution proceeds as fast as the execution unit will allow.

The 8088 and 8086 are completely software compatible by virtue of their identical execution units. Software that is system dependent may not be completely transferable, but software that is not system dependent will operate equally as well on an 8088 and an 8086.

The hardware interface of the 8088 contains the major differences between the two CPUs. The pin assignments are nearly identical, however, with the following functional changes:

- $\text{A}_8\text{--}\text{A}_{15}$ —These pins are only address outputs on the 8088. These address lines are latched internally and remain valid throughout a bus cycle in a manner similar to the 8085 upper address lines.
- $\overline{\text{BHE}}$ has no meaning on the 8088 and has been eliminated.

- \overline{SSO} provides the $\overline{S0}$ status information in the minimum mode. This output occurs on pin 34 in minimum mode only. $\overline{DT/\overline{R}}$, $\overline{IO/\overline{M}}$, and \overline{SSO} provide the complete bus status in minimum mode.
- $\overline{IO/\overline{M}}$ has been inverted to be compatible with the MCS-85 bus structure.
- ALE is delayed by one clock cycle in the minimum mode when entering HALT, to allow the status to be latched with ALE.



231456-10

Figure 10. Medium Complexity System Timing

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias 0°C to +70°C
 Case Temperature (Plastic) 0°C to +95°C
 Case Temperature (CERDIP) 0°C to +75°C
 Storage Temperature -65°C to +150°C
 Voltage on Any Pin with
 Respect to Ground -1.0 to +7V
 Power Dissipation 2.5 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

($T_A = 0^\circ\text{C}$ to 70°C , T_{CASE} (Plastic) = 0°C to 95°C , T_{CASE} (CERDIP) = 0°C to 75°C ,
 $T_A = 0^\circ\text{C}$ to 55°C and $T_{\text{CASE}} = 0^\circ\text{C}$ to 75°C for P8088-2 only
 T_A is guaranteed as long as T_{CASE} is not exceeded)

($V_{\text{CC}} = 5\text{V} \pm 10\%$ for 8088, $V_{\text{CC}} = 5\text{V} \pm 5\%$ for 8088-2 and Extended Temperature EXPRESS)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5	+0.8	V	(Note 1)
V_{IH}	Input High Voltage	2.0	$V_{\text{CC}} + 0.5$	V	(Notes 1, 2)
V_{OL}	Output Low Voltage		0.45	V	$I_{\text{OL}} = 2.0\text{ mA}$
V_{OH}	Output High Voltage	2.4		V	$I_{\text{OH}} = -400\ \mu\text{A}$
I_{CC}	8088 Power Supply Current: 8088-2 P8088		340 350 250	mA	$T_A = 25^\circ\text{C}$
I_{LI}	Input Leakage Current		± 10	μA	$0\text{V} \leq V_{\text{IN}} \leq V_{\text{CC}}$ (Note 3)
I_{LO}	Output and I/O Leakage Current		± 10	μA	$0.45\text{V} \leq V_{\text{OUT}} \leq V_{\text{CC}}$
V_{CL}	Clock Input Low Voltage	-0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.9	$V_{\text{CC}} + 1.0$	V	
C_{IN}	Capacitance If Input Buffer (All Input Except AD ₀ -AD ₇ , RQ/GT)		15	pF	$f_c = 1\text{ MHz}$
C_{IO}	Capacitance of I/O Buffer AD ₀ -AD ₇ , RQ/GT)		15	pF	$f_c = 1\text{ MHz}$

NOTES:

- V_{IL} tested with MN/ $\overline{\text{MX}}$ Pin = 0V
 V_{IH} tested with MN/ $\overline{\text{MX}}$ Pin = 5V
 MN/ $\overline{\text{MX}}$ Pin is a strap Pin
- Not applicable to RQ/GT0 and RQ/GT1 Pins (Pins 30 and 31)
- HOLD and HLDA I_{LI} Min = 30 μA , Max = 500 μA

A.C. CHARACTERISTICS

($T_A = 0^\circ\text{C}$ to 70°C , T_{CASE} (Plastic) = 0°C to 95°C , T_{CASE} (CERDIP) = 0°C to 75°C ,
 $T_A = 0^\circ\text{C}$ to 55°C and $T_{\text{CASE}} = 0^\circ\text{C}$ to 80°C for P8088-2 only
 T_A is guaranteed as long as T_{CASE} is not exceeded)

($V_{\text{CC}} = 5\text{V} \pm 10\%$ for 8088, $V_{\text{CC}} = 5\text{V} \pm 5\%$ for 8088-2 and Extended Temperature EXPRESS)

MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

Symbol	Parameter	8088		8088-2		Units	Test Conditions
		Min	Max	Min	Max		
TCLCL	CLK Cycle Period	200	500	125	500	ns	
TCLCH	CLK Low Time	118		68		ns	
TCHCL	CLK High Time	69		44		ns	
TCH1CH2	CLK Rise Time		10		10	ns	From 1.0V to 3.5V
TCL2CL2	CLK Fall Time		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		20		ns	
TCLDX	Data in Hold Time	10		10		ns	
TR1VCL	RDY Setup Time into 8284 (Notes 1, 2)	35		35		ns	
TCLR1X	RDY Hold Time into 8284 (Notes 1, 2)	0		0		ns	
TRYHCH	READY Setup Time into 8088	118		68		ns	
TCHRYX	READY Hold Time into 8088	30		20		ns	
TRYLCL	READY Inactive to CLK (Note 3)	-8		-8		ns	
THVCH	HOLD Setup Time	35		20		ns	
TINVCH	INTR, NMI, $\overline{\text{TEST}}$ Setup Time (Note 2)	30		15		ns	
TILIH	Input Rise Time (Except CLK)		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12	ns	From 2.0V to 0.8V

A.C. CHARACTERISTICS (Continued)

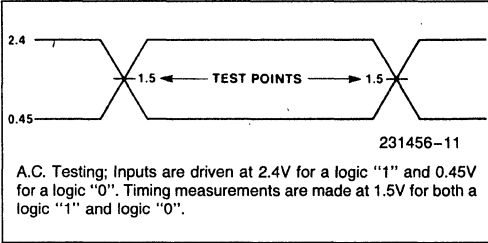
TIMING RESPONSES

Symbol	Parameter	8088		8088-2		Units	Test Conditions
		Min	Max	Min	Max		
TCLAV	Address Valid Delay	10	110	10	60	ns	
TCLAX	Address Hold Time	10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	ns	
TLHLL	ALE Width	TCLCH - 20		TCLCH - 10		ns	
TCLLH	ALE Active Delay		80		50	ns	
TCHLL	ALE Inactive Delay		85		55	ns	
TLLAX	Address Hold Time to ALE Inactive	TCHCL - 10		TCHCL - 10		ns	
TCLDV	Data Valid Delay	10	110	10	60	ns	
TCHDX	Data Hold Time	10		10		ns	
TWHDX	Data Hold Time after \overline{WR}	TCLCH - 30		TCLCH - 30		ns	
TCVCTV	Control Active Delay 1	10	110	10	70	ns	
TCHCTV	Control Active Delay 2	10	110	10	60	ns	
TCVCTX	Control Inactive Delay	10	110	10	70	ns	
TAZRL	Address Float to READ Active	0		0		ns	
TCLRL	\overline{RD} Active Delay	10	165	10	100	ns	
TCLRH	\overline{RD} Inactive Delay	10	150	10	80	ns	
TRHAV	\overline{RD} Inactive to Next Address Active	TCLCL - 45		TCLCL - 40		ns	
TCLHAV	HLDA Valid Delay	10	160	10	100	ns	
TRLRH	\overline{RD} Width	2TCLCL - 75		2TCLCL - 50		ns	
TWLWH	\overline{WR} Width	2TCLCL - 60		2TCLCL - 40		ns	
TAVAL	Address Valid to ALE Low	TCLCH - 60		TCLCH - 40		ns	
TOLOH	Output Rise Time		20		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12	ns	From 2.0V to 0.8V

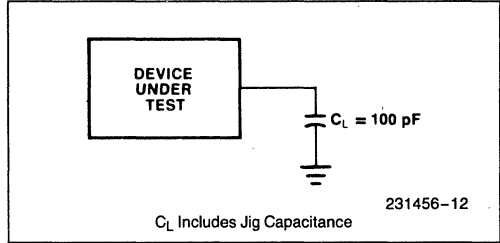
NOTES:

1. Signal at 8284A shown for reference only. See 8284A data sheet for the most recent specifications.
2. Set up requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T2 state (8 ns into T3 state).

A.C. TESTING INPUT, OUTPUT WAVEFORM

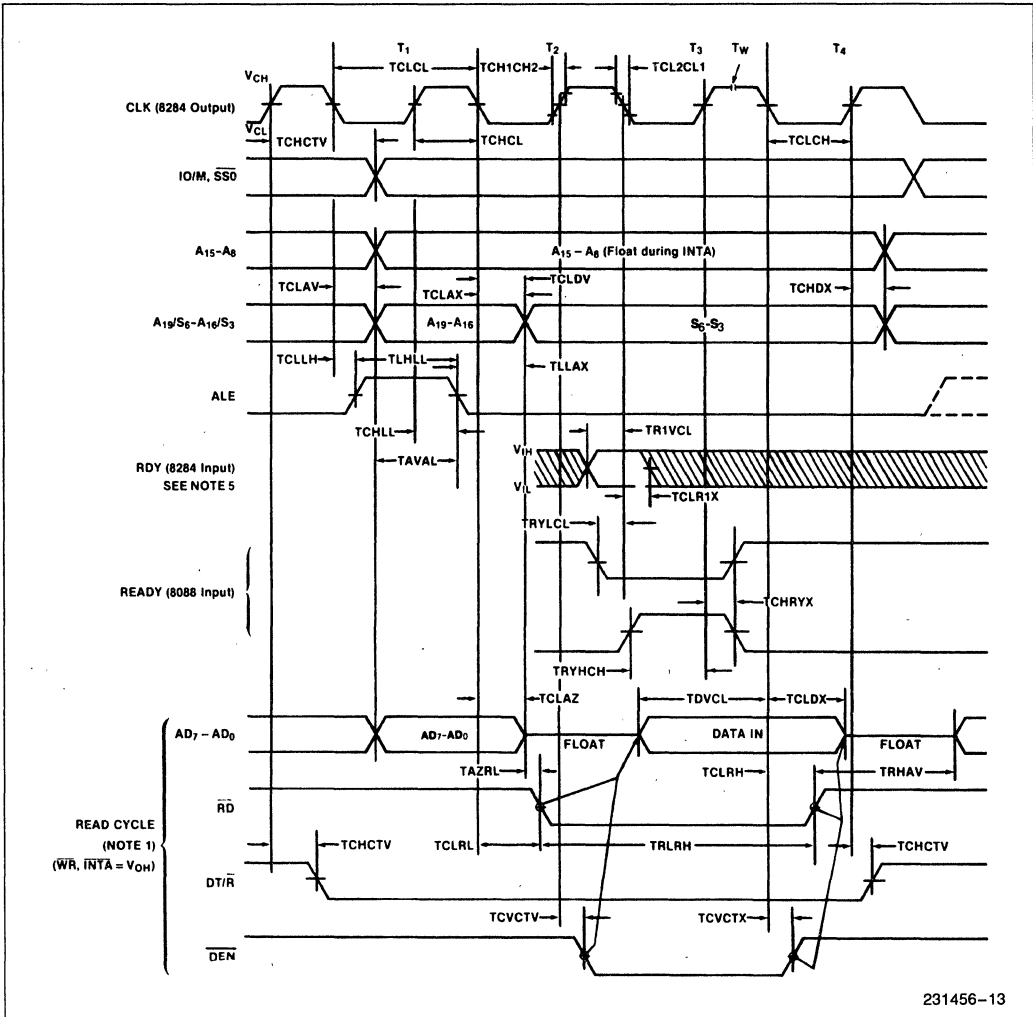


A.C. TESTING LOAD CIRCUIT



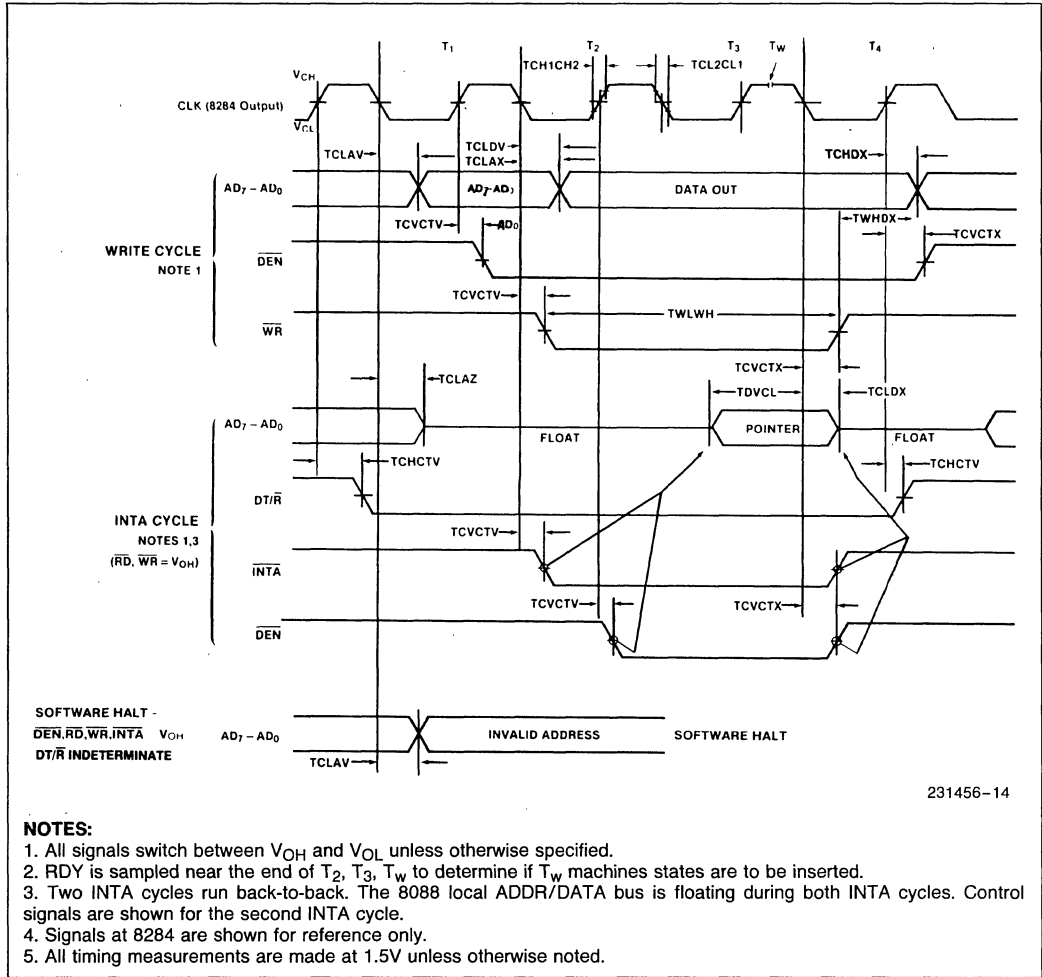
WAVEFORMS

BUS TIMING—MINIMUM MODE SYSTEM



WAVEFORMS (Continued)

BUS TIMING—MINIMUM MODE SYSTEM (Continued)



NOTES:

1. All signals switch between V_{OH} and V_{OL} unless otherwise specified.
2. RDY is sampled near the end of T₂, T₃, T_w to determine if T_w machines states are to be inserted.
3. Two INTA cycles run back-to-back. The 8088 local ADDR/DATA bus is floating during both INTA cycles. Control signals are shown for the second INTA cycle.
4. Signals at 8284 are shown for reference only.
5. All timing measurements are made at 1.5V unless otherwise noted.

A.C. CHARACTERISTICS

MAX MODE SYSTEM (USING 8288 BUS CONTROLLER)

TIMING REQUIREMENTS

Symbol	Parameter	8088		8088-2		Units	Test Conditions	
		Min	Max	Min	Max			
TCLCL	CLK Cycle Period	200	500	125	500	ns		
TCLCH	CLK Low Time	118		68		ns		
TCHCL	CLK High Time	69		44		ns		
TCH1CH2	CLK Rise Time		10		10	ns	From 1.0V to 3.5V	
TCL2CL1	CLK Fall Time		10		10	ns	From 3.5V to 1.0V	
TDVCL	Data in Setup Time	30		20		ns		
TCLDX	Data in Hold Time	10		10		ns		
TR1VCL	RDY Setup Time into 8284 (Notes 1, 2)	35		35		ns		
TCLR1X	RDY Hold Time into 8284 (Notes 1, 2)	0		0		ns		
TRYHCH	READY Setup Time into 8088	118		68		ns		
TCHRYX	READY Hold Time into 8088	30		20		ns		
TRYLCL	READY Inactive to CLK (Note 4)	-8		-8		ns		
TINVCH	Setup Time for Recognition (INTR, NMI, TEST) (Note 2)	30		15		ns		
TGVCH	RQ/GT Setup Time	30		15		ns		
TCHGX	RQ Hold Time into 8088	40		30		ns		
TILIH	Input Rise Time (Except CLK)		20		20	ns		From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12	ns		From 2.0V to 0.8V

A.C. CHARACTERISTICS (Continued)

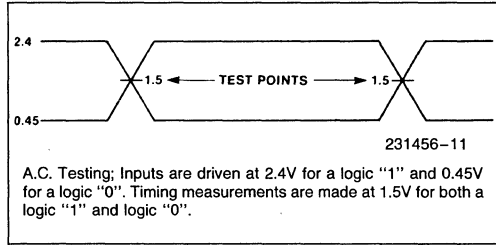
TIMING RESPONSES

Symbol	Parameter	8088		8088-2		Units	Test Conditions	
		Min	Max	Min	Max			
TCLML	Command Active Delay (Note 1)	10	35	10	35	ns	C _L = 20–100 pF for All 8088 Outputs in Addition to Internal Loads	
TCLMH	Command Inactive Delay (Note 1)	10	35	10	35	ns		
TRYHSH	READY Active to Status Passive (Note 3)		110		65	ns		
TCHSV	Status Active Delay	10	110	10	60	ns		
TCLSH	Status Inactive Delay	10	130	10	70	ns		
TCLAV	Address Valid Delay	10	110	10	60	ns		
TCLAX	Address Hold Time	10		10		ns		
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	ns		
TSVLH	Status Valid to ALE High (Note 1)		15		15	ns		
TSMVCH	Status Valid to MCE High (Note 1)		15		15	ns		
TCLLH	CLK Low to ALE Valid (Note 1)		15		15	ns		
TCLMCH	CLK Low to MCE (Note 1)		15		15	ns		
TCHLL	ALE Inactive Delay (Note 1)		15		15	ns		
TCLMCL	MCE Inactive Delay (Note 1)		15		15	ns		
TCLDV	Data Valid Delay	10	110	10	60	ns		
TCHDX	Data Hold Time	10		10		ns		
TCVNV	Control Active Delay (Note 1)	5	45	5	45	ns		
TCVNX	Control Inactive Delay (Note 1)	10	45	10	45	ns		
TAZRL	Address Float to Read Active	0		0		ns		
TCLRL	\overline{RD} Active Delay	10	165	10	100	ns		
TCLRH	\overline{RD} Inactive Delay	10	150	10	80	ns		
TRHAV	\overline{RD} Inactive to Next Address Active	TCLCL – 45		TCLCL – 40		ns		
TCHDTL	Direction Control Active Delay (Note 1)		50		50	ns		
TCHDTH	Direction Control Inactive Delay (Note 1)		30		30	ns		
TCLGL	GT Active Delay		85		50	ns		
TCLGH	GT Inactive Delay		85		50	ns		
TRLRH	\overline{RD} Width	2TCLCL – 75		2TCLCL – 50		ns		
TOLOH	Output Rise Time		20		20	ns		From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12	ns		From 2.0V to 0.8V

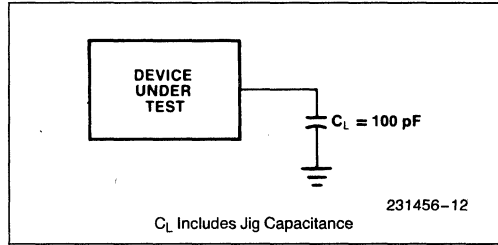
NOTES:

- Signal at 8284 or 8288 shown for reference only.
- Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
- Applies only to T3 and wait states.
- Applies only to T2 state (8 ns into T3 state).

A.C. TESTING INPUT, OUTPUT WAVEFORM

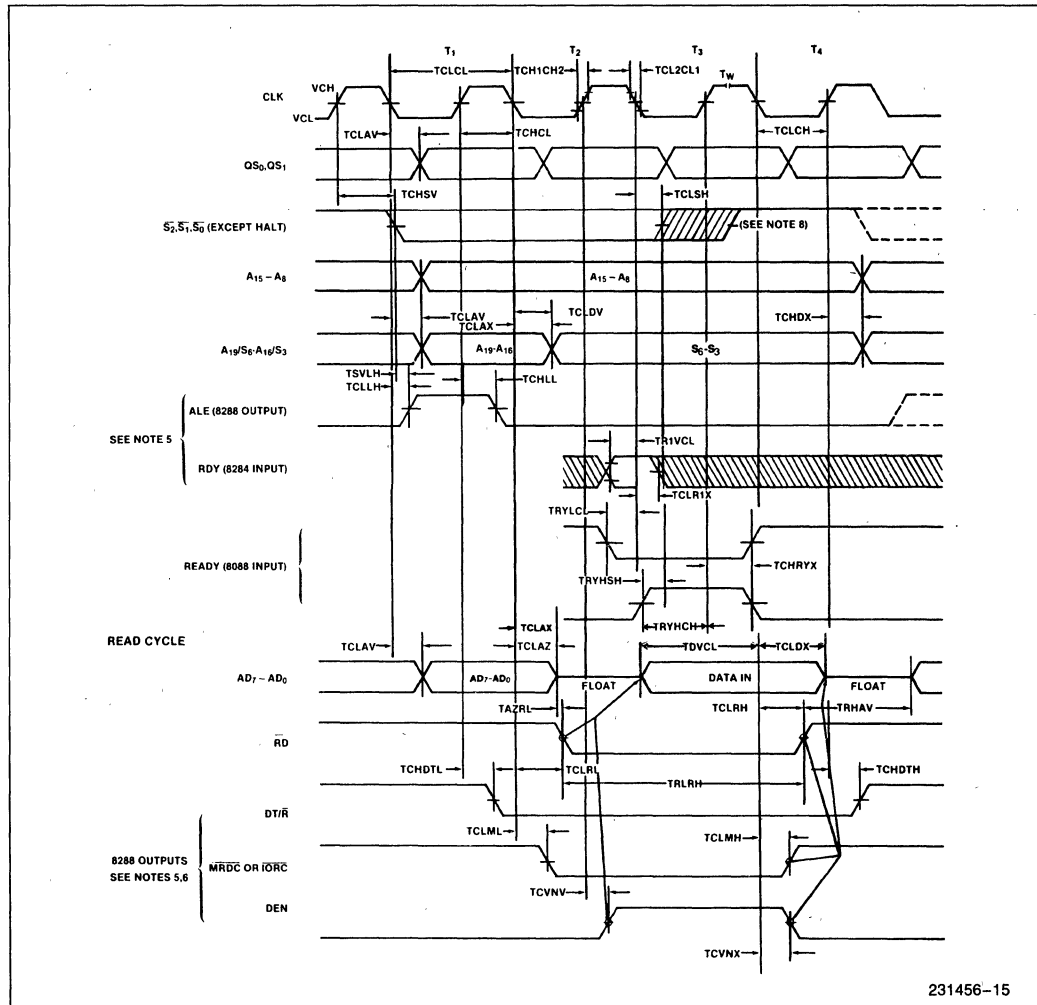


A.C. TESTING LOAD CIRCUIT



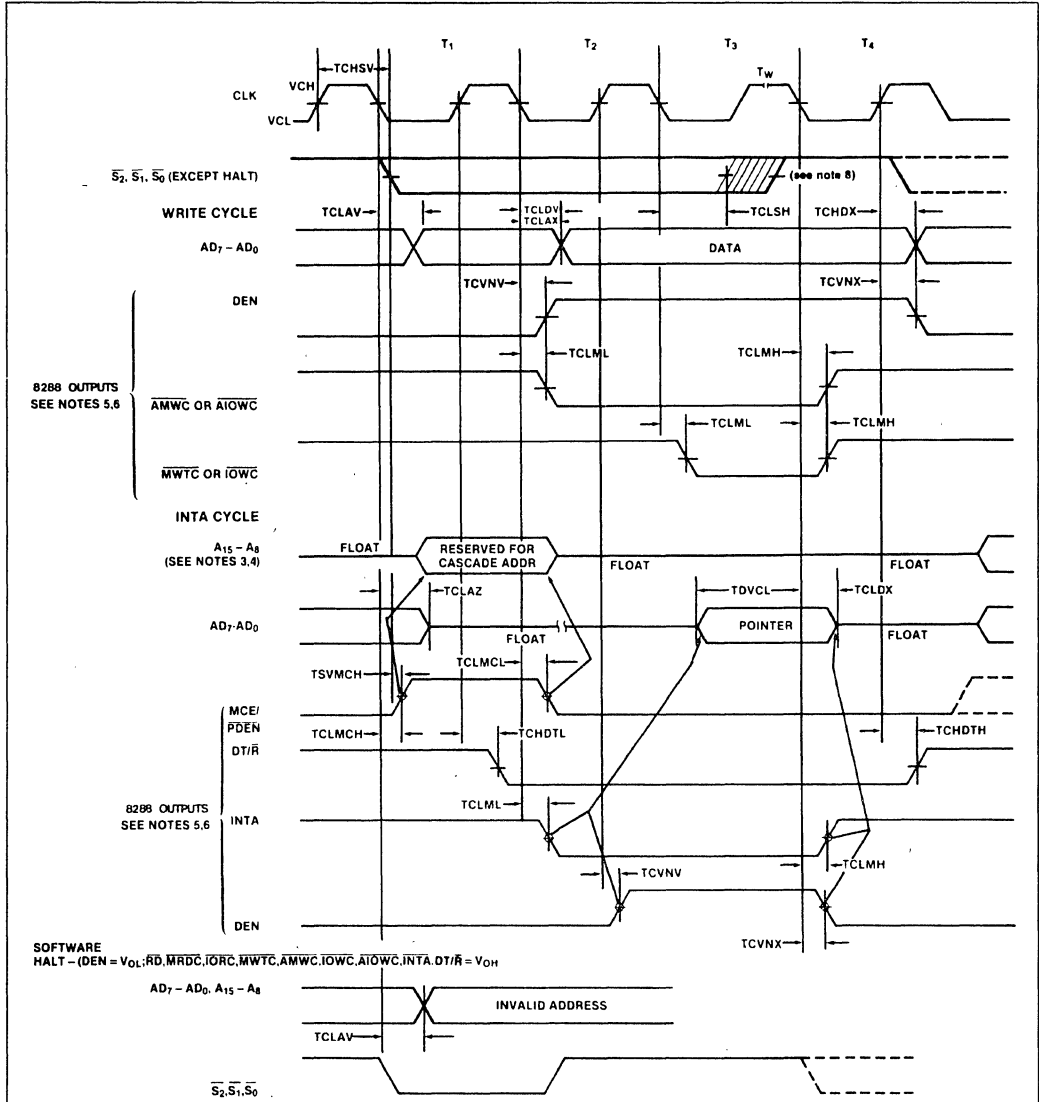
WAVEFORMS (Continued)

BUS TIMING—MAXIMUM MODE SYSTEM



WAVEFORMS (Continued)

BUS TIMING—MAXIMUM MODE SYSTEM (USING 8288)



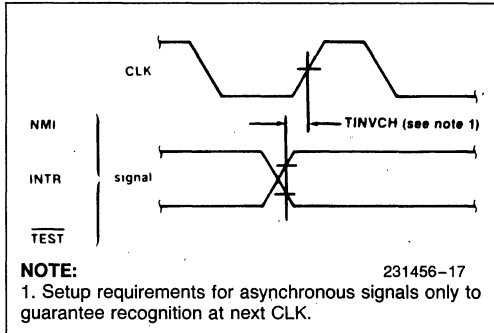
NOTES:

1. All signals switch between V_{0H} and V_{0L} unless otherwise specified.
2. RDY is sampled near the end of T₂, T₃, T_w to determine if T_w machine states are to be inserted.
3. Cascade address is valid between first and second INTA cycles.
4. Two INTA cycles run back-to-back. The 8088 local ADDR/DATA bus is floating during both INTA cycles. Control for pointer address is shown for second INTA cycle.
5. Signals at 8284 or 8288 are shown for reference only.
6. The issuance of the 8288 command and control signals (MRDC, MWTC, AMWC, IORC, IOWC, A₁₀WC, INTA and DEN) lags the active high 8288 CEN.
7. All timing measurements are made at 1.5V unless otherwise noted.
8. Status inactive in state just prior to T₄.

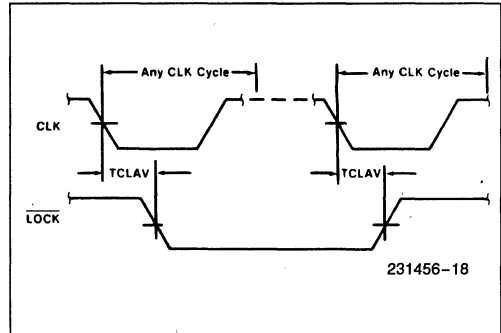
231456-16

WAVEFORMS (Continued)

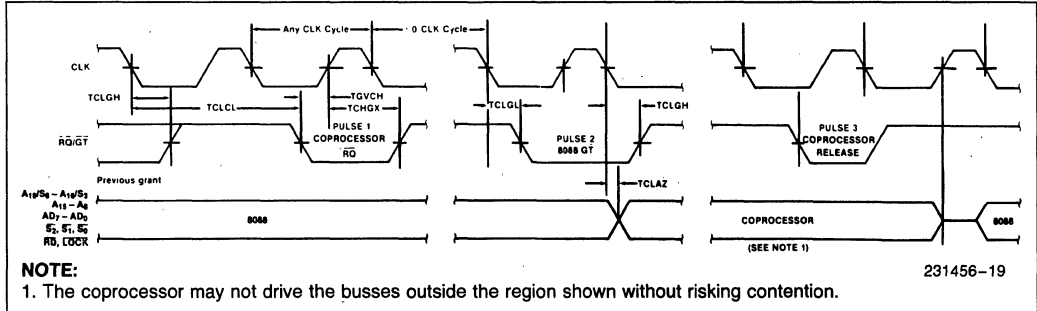
ASYNCHRONOUS SIGNAL RECOGNITION



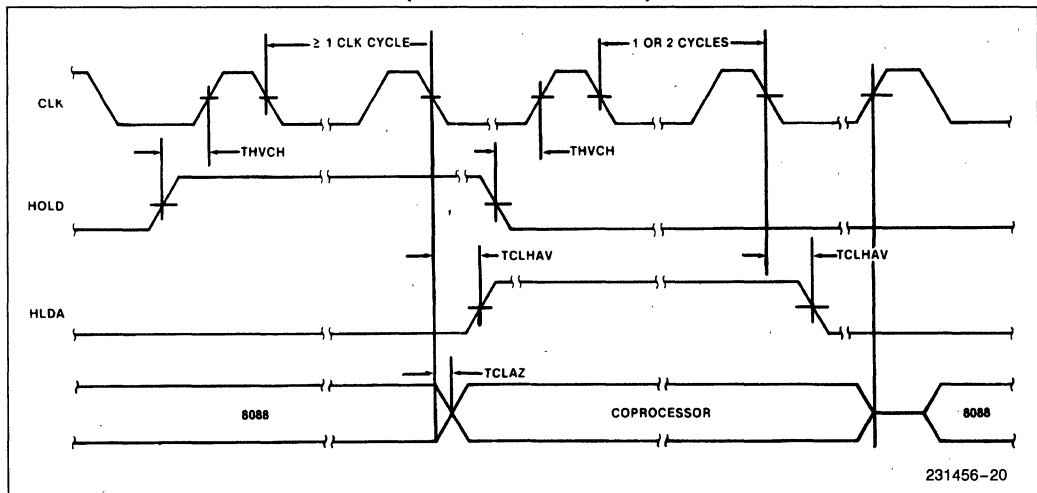
BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)



REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)



HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)



8086/8088 Instruction Set Summary

Mnemonic and Description	Instruction Code			
DATA TRANSFER				
MOV = Move:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Register/Memory to/from Register	1 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate to Register	1 0 1 1 w reg	data	data if w = 1	
Memory to Accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high	
Accumulator to Memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/Memory to Segment Register	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment Register to Register/Memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		
PUSH = Push:				
Register/Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m		
Register	0 1 0 1 0 reg			
Segment Register	0 0 0 reg 1 1 0			
POP = Pop:				
Register/Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m		
Register	0 1 0 1 1 reg			
Segment Register	0 0 0 reg 1 1 1			
XCHG = Exchange:				
Register/Memory with Register	1 0 0 0 0 1 1 w	mod reg r/m		
Register with Accumulator	1 0 0 1 0 reg			
IN = Input from:				
Fixed Port	1 1 1 0 0 1 0 w	port		
Variable Port	1 1 1 0 1 1 0 w			
OUT = Output to:				
Fixed Port	1 1 1 0 0 1 1 w	port		
Variable Port	1 1 1 0 1 1 1 w			
XLAT = Translate Byte to AL	1 1 0 1 0 1 1 1			
LEA = Load EA to Register	1 0 0 0 1 1 0 1	mod reg r/m		
LDS = Load Pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m		
LES = Load Pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m		
LAHF = Load AH with Flags	1 0 0 1 1 1 1 1			
SAHF = Store AH into Flags	1 0 0 1 1 1 1 0			
PUSHF = Push Flags	1 0 0 1 1 1 0 0			
POPF = Pop Flags	1 0 0 1 1 1 0 1			

8086/8088 Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code			
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
ARITHMETIC				
ADD = Add:				
Reg./Memory with Register to Either	0 0 0 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s:w = 01
Immediate to Accumulator	0 0 0 0 0 1 0 w	data	data if w = 1	
ADC = Add with Carry:				
Reg./Memory with Register to Either	0 0 0 1 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s:w = 01
Immediate to Accumulator	0 0 0 1 0 1 0 w	data	data if w = 1	
INC = Increment:				
Register/Memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m		
Register	0 1 0 0 0 reg			
AAA = ASCII Adjust for Add	0 0 1 1 0 1 1 1			
BAA = Decimal Adjust for Add	0 0 1 0 0 1 1 1			
SUB = Subtract:				
Reg./Memory and Register to Either	0 0 1 0 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s:w = 01
Immediate from Accumulator	0 0 1 0 1 1 0 w	data	data if w = 1	
SSB = Subtract with Borrow				
Reg./Memory and Register to Either	0 0 0 1 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s:w = 01
Immediate from Accumulator	0 0 0 1 1 1 w	data	data if w = 1	
DEC = Decrement:				
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m		
Register	0 1 0 0 1 reg			
NEG = Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		
CMP = Compare:				
Register/Memory and Register	0 0 1 1 1 0 d w	mod reg r/m		
Immediate with Register/Memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s:w = 01
Immediate with Accumulator	0 0 1 1 1 1 0 w	data	data if w = 1	
AAS = ASCII Adjust for Subtract	0 0 1 1 1 1 1 1			
DAS = Decimal Adjust for Subtract	0 0 1 0 1 1 1 1			
MUL = Multiply (Unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m		
IMUL = Integer Multiply (Signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m		
AAM = ASCII Adjust for Multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		
DIV = Divide (Unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m		
IDIV = Integer Divide (Signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m		
AAD = ASCII Adjust for Divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		
CBW = Convert Byte to Word	1 0 0 1 1 0 0 0			
CWD = Convert Word to Double Word	1 0 0 1 1 0 0 1			

8086/8088 Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code			
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
LOGIC				
NOT = Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m		
SHL/SAL = Shift Logical/Arithmetic Left	1 1 0 1 0 0 v w	mod 1 0 0 r/m		
SHR = Shift Logical Right	1 1 0 1 0 0 v w	mod 1 0 1 r/m		
SAR = Shift Arithmetic Right	1 1 0 1 0 0 v w	mod 1 1 1 r/m		
ROL = Rotate Left	1 1 0 1 0 0 v w	mod 0 0 0 r/m		
ROR = Rotate Right	1 1 0 1 0 0 v w	mod 0 0 1 r/m		
RCL = Rotate Through Carry Flag Left	1 1 0 1 0 0 v w	mod 0 1 0 r/m		
RCR = Rotate Through Carry Right	1 1 0 1 0 0 v w	mod 0 1 1 r/m		
AND = And:				
Reg./Memory and Register to Either	0 0 1 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w = 1
Immediate to Accumulator	0 0 1 0 0 1 0 w	data	data if w = 1	
TEST = And Function to Flags. No Result:				
Register/Memory and Register	1 0 0 0 0 1 0 w	mod reg r/m		
Immediate Data and Register/Memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate Data and Accumulator	1 0 1 0 1 0 0 w	data	data if w = 1	
OR = Or:				
Reg./Memory and Register to Either	0 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1
Immediate to Accumulator	0 0 0 0 1 1 0 w	data	data if w = 1	
XOR = Exclusive or:				
Reg./Memory and Register to Either	0 0 1 1 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1
Immediate to Accumulator	0 0 1 1 0 1 0 w	data	data if w = 1	
STRING MANIPULATION				
REP = Repeat	1 1 1 1 0 0 1 z			
MOVS = Move Byte/Word	1 0 1 0 0 1 0 w			
CMPS = Compare Byte/Word	1 0 1 0 0 1 1 w			
SCAS = Scan Byte/Word	1 0 1 0 1 1 1 w			
LODS = Load Byte/Wd to AL/AX	1 0 1 0 1 1 0 w			
STOS = Stor Byte/Wd from AL/A	1 0 1 0 1 0 1 w			
CONTROL TRANSFER				
CALL = Call:				
Direct Within Segment	1 1 1 0 1 0 0 0	disp-low	disp-high	
Indirect Within Segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m		
Direct Intersegment	1 0 0 1 1 0 1 0	offset-low	offset-high	
		seg-low	seg-high	
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m		

8086/8088 Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code		
JMP = Unconditional Jump:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Direct Within Segment	1 1 1 0 1 0 0 1	disp-low	disp-high
Direct Within Segment-Short	1 1 1 0 1 0 1 1	disp	
Indirect Within Segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	
Direct Intersegment	1 1 1 0 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	
RET = Return from CALL:			
Within Segment	1 1 0 0 0 0 1 1		
Within Seg Adding Immed to SP	1 1 0 0 0 0 1 0	data-low	data-high
Intersegment	1 1 0 0 1 0 1 1		
Intersegment Adding Immediate to SP	1 1 0 0 1 0 1 0	data-low	data-high
JE/JZ = Jump on Equal/Zero	0 1 1 1 0 1 0 0	disp	
JL/JNGE = Jump on Less/Not Greater or Equal	0 1 1 1 1 1 0 0	disp	
JLE/JNG = Jump on Less or Equal/Not Greater	0 1 1 1 1 1 1 0	disp	
JB/JNAE = Jump on Below/Not Above or Equal	0 1 1 1 0 0 1 0	disp	
JBE/JNA = Jump on Below or Equal/Not Above	0 1 1 1 0 1 1 0	disp	
JP/JPE = Jump on Parity/Parity Even	0 1 1 1 1 0 1 0	disp	
JO = Jump on Overflow	0 1 1 1 0 0 0 0	disp	
JS = Jump on Sign	0 1 1 1 1 0 0 0	disp	
JNE/JNZ = Jump on Not Equal/Not Zero	0 1 1 1 0 1 0 1	disp	
JNL/JGE = Jump on Not Less/Greater or Equal	0 1 1 1 1 1 0 1	disp	
JNLE/JG = Jump on Not Less or Equal/Greater	0 1 1 1 1 1 1 1	disp	
JNB/JAE = Jump on Not Below/Above or Equal	0 1 1 1 0 0 1 1	disp	
JNBE/JA = Jump on Not Below or Equal/Above	0 1 1 1 0 1 1 1	disp	
JNP/JPO = Jump on Not Par/Par Odd	0 1 1 1 1 0 1 1	disp	
JNO = Jump on Not Overflow	0 1 1 1 0 0 0 1	disp	
JNS = Jump on Not Sign	0 1 1 1 1 0 0 1	disp	
LOOP = Loop CX Times	1 1 1 0 0 0 1 0	disp	
LOOPZ/LOOPE = Loop While Zero/Equal	1 1 1 0 0 0 0 1	disp	
LOOPNZ/LOOPNE = Loop While Not Zero/Equal	1 1 1 0 0 0 0 0	disp	
JCZX = Jump on CX Zero	1 1 1 0 0 0 1 1	disp	
INT = Interrupt			
Type Specified	1 1 0 0 1 1 0 1	type	
Type 3	1 1 0 0 1 1 0 0		
INTO = Interrupt on Overflow	1 1 0 0 1 1 1 0		
IRET = Interrupt Return	1 1 0 0 1 1 1 1		

8086/8088 Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code	
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
PROCESSOR CONTROL		
CLC = Clear Carry	1 1 1 1 1 0 0 0	
CMC = Complement Carry	1 1 1 1 0 1 0 1	
STC = Set Carry	1 1 1 1 1 0 0 1	
CLD = Clear Direction	1 1 1 1 1 1 0 0	
STD = Set Direction	1 1 1 1 1 1 0 1	
CLI = Clear Interrupt	1 1 1 1 1 0 1 0	
STI = Set Interrupt	1 1 1 1 1 0 1 1	
HLT = Halt	1 1 1 1 0 1 0 0	
WAIT = Wait	1 0 0 1 1 0 1 1	
ESC = Escape (to External Device)	1 1 0 1 1 x x x	mod x x x r/m
LOCK = Bus Lock Prefix	1 1 1 1 0 0 0 0	

NOTES:

AL = 8-bit accumulator
 AX = 16-bit accumulator
 CX = Count register
 DS = Data segment
 ES = Extra segment
 Above/below refers to unsigned value
 Greater = more positive;
 Less = less positive (more negative) signed values
 if d = 1 then "to" reg; if d = 0 then "from" reg
 if w = 1 then word instruction; if w = 0 then byte instruction
 if mod = 11 then r/m is treated as a REG field
 if mod = 00 then DISP = 0*, disp-low and disp-high are absent
 if mod = 01 then DISP = disp-low sign-extended to 16 bits, disp-high is absent
 if mod = 10 then DISP = disp-high; disp-low
 if r/m = 000 then EA = (BX) + (SI) + DISP
 if r/m = 001 then EA = (BX) + (DI) + DISP
 if r/m = 010 then EA = (BP) + (SI) + DISP
 if r/m = 011 then EA = (BP) + (DI) + DISP
 if r/m = 100 then EA = (SI) + DISP
 if r/m = 101 then EA = (DI) + DISP
 if r/m = 110 then EA = (BP) + DISP*
 if r/m = 111 then EA = (BX) + DISP
 DISP follows 2nd byte of instruction (before data if required)
 *except if mod = 00 and r/m = then EA = disp-high; disp-low.
 if s:w = 01 then 16 bits of immediate data form the operand
 if s:w = 11 then an immediate data byte is sign extended to form the 16-bit operand
 if v = 0 then "count" = 1; if v = 1 then "count" in (CL) register
 x = don't care
 z is used for string primitives for comparison with ZF FLAG
SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:

FLAGS =
 X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

Mnemonics © Intel, 1978

DATA SHEET REVISION REVIEW

The following list represents key differences between this and the -004 data sheet. Please review this summary carefully.

1. The Pin Description Table has been modified to indicate that the HOLD and HLDA pins both have internal pull-up resistors. The input leakage current (I_{LI}) in the D.C. Characteristics section has been modified for these pins.

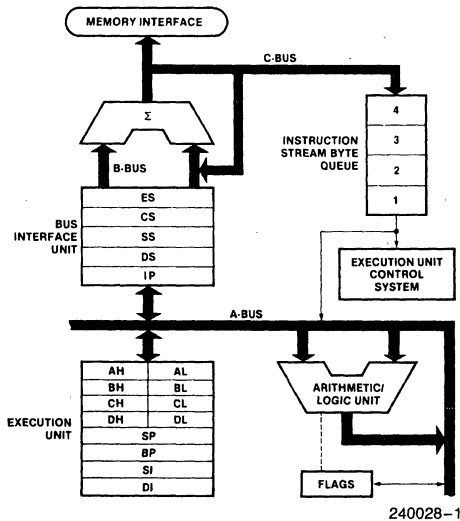


80C88A

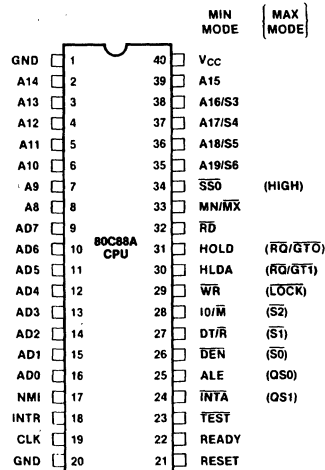
8-BIT CHMOS MICROPROCESSOR

- Pin-for-Pin and Functionally Compatible to Industry Standard HMOS 8088
- Direct Software Compatibility with 80C86, 8086, 8088
- Fully Static Design with Frequency Range from D.C. to:
 - 8 MHz for 80C88A-2
- Low Power Operation
 - Operating $I_{CC} = 10 \text{ mA/MHz}$
 - Standby $I_{CCs} = 500 \mu\text{A max}$
- Bus-Hold Circuitry Eliminates Pull-Up Resistors
- Direct Addressing Capability of 1 MByte of Memory
- Architecture Designed for Powerful Assembly Language and Efficient High Level Languages
- 24 Operand Addressing Modes
- Byte, Word and Block Operations
- 8 and 16-Bit Signed and Unsigned Arithmetic
 - Binary or Decimal
 - Multiply and Divide
- Available in 40-Lead Plastic DIP
 - (See Packaging Spec., Order # 231369)

The Intel 80C88A is a high performance, CHMOS version of the industry standard HMOS 8088 8-bit CPU. The processor has attributes of both 8 and 16-bit microprocessors. The 80C88A, available in 8 MHz clock rate, offers two modes of operation: MINimum for small systems and MAXimum for larger applications such as multi-processing. It is available in 40-pin DIP.



**Figure 1. 80C88A CPU
Functional Block Diagram**



**Figure 2. 80C88A 40-Lead
DIP Configuration**

Table 1. Pin Description

The following pin function descriptions are for 80C88A systems in either minimum or maximum mode. The "local bus" in these descriptions is the direct multiplexed bus interface connection to the 80C88A (without regard to additional bus buffers).

Symbol	Pin No.	Type	Name and Function																		
AD7-AD0	9-16	I/O	ADDRESS DATA BUS: These lines constitute the time multiplexed memory/I/O address (T1) and data (T2, T3, Tw, and T4) bus. These lines are active HIGH and float to 3-state OFF ⁽¹⁾ during interrupt acknowledge and local bus "hold acknowledge".																		
A15-A8	2-8, 39	O	ADDRESS BUS: These lines provide address bits 8 through 15 for the entire bus cycle (T1-T4). These lines do not have to be latched by ALE to remain valid. A15-A8 are active HIGH and float to 3-state OFF ⁽¹⁾ during interrupt acknowledge and local bus "hold acknowledge".																		
A19/S6, A18/S5, A17/S4, A16/S3	35-38	O	<p>ADDRESS/STATUS: During T1, these are the four most significant address lines for memory operations. During I/O operations, these lines are LOW. During memory and I/O operations, status information is available on these lines during T2, T3, Tw, and T4. S6 is always low. The status of the interrupt enable flag bit (S5) is updated at the beginning of each clock cycle. S4 and S3 are encoded as shown.</p> <p>This information indicates which segment register is presently being used for data accessing.</p> <p>These lines float to 3-state OFF⁽¹⁾ during local bus "hold acknowledge".</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 33%;">S4</th> <th style="width: 33%;">S3</th> <th style="width: 33%;">CHARACTERISTICS</th> </tr> </thead> <tbody> <tr> <td>0 (LOW)</td> <td>0</td> <td>Alternate Data</td> </tr> <tr> <td>0</td> <td>1</td> <td>Stack</td> </tr> <tr> <td>1 (HIGH)</td> <td>0</td> <td>Code or None</td> </tr> <tr> <td>1</td> <td>1</td> <td>Data</td> </tr> <tr> <td colspan="3">S6 is 0 (LOW)</td> </tr> </tbody> </table>	S4	S3	CHARACTERISTICS	0 (LOW)	0	Alternate Data	0	1	Stack	1 (HIGH)	0	Code or None	1	1	Data	S6 is 0 (LOW)		
S4	S3	CHARACTERISTICS																			
0 (LOW)	0	Alternate Data																			
0	1	Stack																			
1 (HIGH)	0	Code or None																			
1	1	Data																			
S6 is 0 (LOW)																					
\overline{RD}	32	O	<p>READ: Read strobe indicates that the processor is performing a memory or I/O read cycle, depending on the state of the IO/\overline{M} pin or S2. This signal is used to read devices which reside on the 80C88A local bus. \overline{RD} is active LOW during T2, T3 and Tw of any read cycle, and is guaranteed to remain HIGH in T2 until the 80C88A local bus has floated.</p> <p>This signal floats to 3-state OFF⁽¹⁾ in "hold acknowledge".</p>																		
READY	22	I	READY: is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The RDY signal from memory or I/O is synchronized by the 82C84A clock generator to form READY. This signal is active HIGH. The 80C88A READY input is not synchronized. Correct operation is not guaranteed if the set up and hold times are not met.																		

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
INTR	18	I	INTERRUPT REQUEST: is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.
$\overline{\text{TEST}}$	23	I	$\overline{\text{TEST}}$: input is examined by the "wait for test" instruction. If the $\overline{\text{TEST}}$ input is LOW, execution continues, otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.
NMI	17	I	NON-MASKABLE INTERRUPT: is an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from a LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.
RESET	21	I	RESET: causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the instruction set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	CLOCK: provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V _{CC}	40		V_{CC}: is the +5V ± 10% power supply pin.
GND	1, 20		GND: are the ground pins. Both must be connected.
MN/ $\overline{\text{MX}}$	33	I	MINIMUM/MAXIMUM: indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

The following pin function descriptions are for the 80C88A minimum mode (i.e., MN/ $\overline{\text{MX}}$ = V_{CC}). Only the pin functions which are unique to minimum mode are described; all other pin functions are as described above.

IO/ $\overline{\text{M}}$	28	O	STATUS LINE: is an inverted maximum mode $\overline{\text{S2}}$. It is used to distinguish a memory access from an I/O access. IO/ $\overline{\text{M}}$ becomes valid in the T4 preceding a bus cycle and remains valid until the final T4 of the cycle (I/O = HIGH, M = LOW). IO/ $\overline{\text{M}}$ floats to 3-state OFF ⁽¹⁾ in local bus "hold acknowledge".
WR	29	O	WRITE: strobe indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the IO/ $\overline{\text{M}}$ signal. WR is active for T2, T3, and Tw of any write cycle. It is active LOW, and floats to 3-state OFF ⁽¹⁾ in local bus "hold acknowledge".
$\overline{\text{INTA}}$	24	O	INTA: is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T2, T3, and Tw of each interrupt acknowledge cycle.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function			
ALE	25	O	<p>ADDRESS LATCH ENABLE: is provided by the processor to latch the address into an address latch. It is a HIGH pulse active during clock low of T1 of any bus cycle. Note that ALE is never floated.</p>			
DT/ \bar{R}	27	O	<p>DATA TRANSMIT/RECEIVE: is needed in a minimum system that desires to use a data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically, DT/\bar{R} is equivalent to $\bar{S}1$ in the maximum mode, and its timing is the same as for IO/\bar{M} (T = HIGH, R = LOW). This signal floats to 3-state OFF⁽¹⁾ in local "hold acknowledge".</p>			
\bar{DEN}	26	O	<p>DATA ENABLE: is provided as an output enable for the transceiver in a minimum system which uses the transceiver. \bar{DEN} is active LOW during each memory and I/O access, and for \bar{INTA} cycles. For a read or \bar{INTA} cycle, it is active from the middle of T2 until the middle of T4, while for a write cycle, it is active from the beginning of T2 until the middle of T4. \bar{DEN} floats to 3-state OFF⁽¹⁾ during local bus "hold acknowledge".</p>			
HOLD, HLDA	30, 31	I, O	<p>HOLD: indicates that another master is requesting a local bus "hold". To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement, in the middle of a T4 or T_i clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor lowers HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines.</p> <p>Hold is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the set up time.</p>			
$\bar{SS}0$	34	O	<p>STATUS LINE: is logically equivalent to $\bar{S}0$ in the maximum mode. The combination of $\bar{SS}0$, IO/\bar{M} and DT/\bar{R} allows the system to completely decode the current bus cycle status.</p>			
			IO/ \bar{M}	DT/ \bar{R}	$\bar{SS}0$	CHARACTERISTICS
			1(HIGH)	0	0	Interrupt Acknowledge
			1	0	1	Read I/O port
			1	1	0	Write I/O port
			1	1	1	Halt
			0(LOW)	0	0	Code access
			0	0	1	Read memory
			0	1	0	Write memory
			0	1	1	Passive

Table 1. Pin Description (Continued)

The following pin function descriptions are for the 80C88A/82C88 system in maximum mode (i.e., $MN/\overline{MX} = GND$.) Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

Symbol	Pin No.	Type	Name and Function			
$\overline{S2}, \overline{S1}, \overline{S0}$	26-28	O	<p>STATUS: is active during clock high of T4, T1, and T2, and is returned to the passive state (1,1,1) during T3 or during Tw when READY is HIGH. This status is used by the 82C88 bus controller to generate all memory and I/O access control signals. Any change by $\overline{S2}, \overline{S1},$ or $\overline{S0}$ during T4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T3 or Tw is used to indicate the end of a bus cycle.</p> <p>These signals float to 3-state OFF⁽¹⁾ during "hold acknowledge". During the first clock cycle after RESET becomes active, these signals are active HIGH. After this first clock, they float to 3-state OFF.</p>			
			S2	S1	S0	CHARACTERISTICS
			0 (LOW)	0	0	Interrupt Acknowledge
			0	0	1	Read I/O port
			0	1	0	Write I/O port
			0	1	1	Halt
			1 (HIGH)	0	0	Code access
			1	0	1	Read memory
			1	1	0	Write memory
			1	1	1	Passive
$\overline{RQ}/\overline{GT0},$ $\overline{RQ}/\overline{GT1}$	30, 31	I/O	<p>REQUEST/GRANT: pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT0}$ having higher priority than $\overline{RQ}/\overline{GT1}$. $\overline{RQ}/\overline{GT}$ has an internal pull-up resistor, so may be left unconnected. The request/grant sequence is as follows (see timing diagram):</p> <ol style="list-style-type: none"> 1. A pulse of one CLK wide from another local bus master indicates a local bus request ("hold") to the 80C88A (pulse 1). 2. During a T4 or T1 clock cycle, a pulse one clock wide from the 80C88A to the requesting master (pulse 2), indicates that the 80C88A has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge". The same rules as for HOLD/HOLDA apply as for when the bus is released. 3. A pulse one CLK wide from the requesting master indicates to the 80C88A (pulse 3) that the "hold" request is about to end and that the 80C88A can reclaim the local bus at the next CLK. The CPU then enters T4. 			

Table 1. Pin Descriptions (Continued)

Symbol	Pin No.	Type	Name and Function															
$\overline{RQ/GT0}$, $\overline{RQ/GT1}$	30, 31	I/O	<p>Each master-master exchange of the local bus is a sequence of three pulses. There must be one idle CLK cycle after each bus exchange. Pulses are active LOW.</p> <p>If the request is made while the CPU is performing a memory cycle, it will release the local bus during T4 of the cycle when all the following conditions are met:</p> <ol style="list-style-type: none"> 1. Request occurs on or before T2. 2. Current cycle is not the low bit of a word. 3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence. 4. A locked instruction is not currently executing. <p>If the local bus is idle when the request is made the two possible events will follow:</p> <ol style="list-style-type: none"> 1. Local bus will be released during the next clock. 2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied. 															
\overline{LOCK}	29	O	<p>LOCK: indicates that other system bus masters are not to gain control of the system bus while \overline{LOCK} is active (LOW). The \overline{LOCK} signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state OFF⁽¹⁾ in "hold acknowledge".</p>															
QS1, QS0	24, 25	O	<p>QUEUE STATUS: provide status to allow external tracking of the internal 80C88A instruction queue.</p> <p>The queue status is valid during the CLK cycle after which the queue operation is performed.</p> <table border="1"> <thead> <tr> <th>QS1</th> <th>QS0</th> <th>CHARACTERISTICS</th> </tr> </thead> <tbody> <tr> <td>0(LOW)</td> <td>0</td> <td>No operation</td> </tr> <tr> <td>0</td> <td>1</td> <td>First byte of opcode from queue</td> </tr> <tr> <td>1(HIGH)</td> <td>0</td> <td>Empty the queue</td> </tr> <tr> <td>1</td> <td>1</td> <td>Subsequent byte from queue</td> </tr> </tbody> </table>	QS1	QS0	CHARACTERISTICS	0(LOW)	0	No operation	0	1	First byte of opcode from queue	1(HIGH)	0	Empty the queue	1	1	Subsequent byte from queue
QS1	QS0	CHARACTERISTICS																
0(LOW)	0	No operation																
0	1	First byte of opcode from queue																
1(HIGH)	0	Empty the queue																
1	1	Subsequent byte from queue																
—	34	O	Pin 34 is always high in the maximum mode.															

NOTE:

1. See the section on Bus Hold Circuitry.

FUNCTIONAL DESCRIPTION

STATIC OPERATION

All 80C88A circuitry is of static design. Internal registers, counters and latches are static and require no refresh as with dynamic circuit design. This eliminates the minimum operating frequency restriction placed on other microprocessors. The CMOS 80C88A can operate from DC to the appropriate upper frequency limit. The processor clock may be stopped in either state (high/low) and held there indefinitely. This type of operation is especially useful for system debug or power critical applications.

The 80C88A can be single stepped using only the CPU clock. This state can be maintained as long as is necessary. Single step clock operation allows simple interface circuitry to provide critical information for bringing up your system.

Static design also allows very low frequency operation. In a power critical situation, this can provide extremely low power operation since 80C88A power dissipation is directly related to operating frequency. As the system frequency is reduced, so is the operating power until ultimately, at a DC input frequency, the 80C88A power requirement is the standby current.

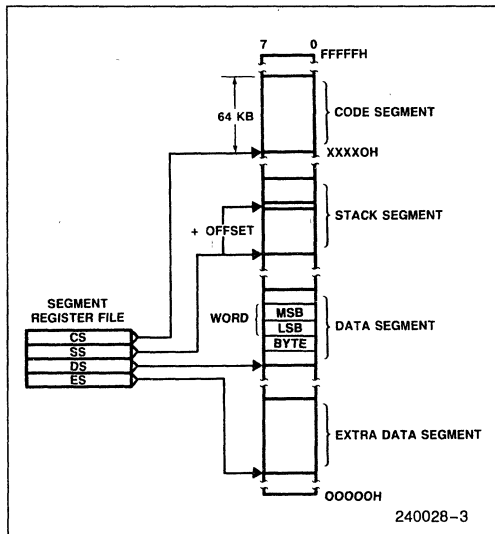


Figure 3. Memory Organization

MEMORY ORGANIZATION

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64K bytes each, with each segment falling on 16-byte boundaries. (See Figure 3.)

All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the addressing needs of programs. The segment register to be selected is automatically chosen according to the rules of the following table. All information in one segment type share the same logical attributes (e.g. code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster, and more structured.

Word (16-bit) operands can be located on even or odd address boundaries. For address and data operands, the least significant byte of the word is stored in the lower valued address location and the most significant byte in the next higher address location. The BIU will automatically execute two fetch or write cycles for 16-bit operands.

Certain locations in memory are reserved for specific CPU operations. (See Figure 4.) Locations from addresses FFFF0H through FFFFFH are reserved for operations including a jump to the initial system

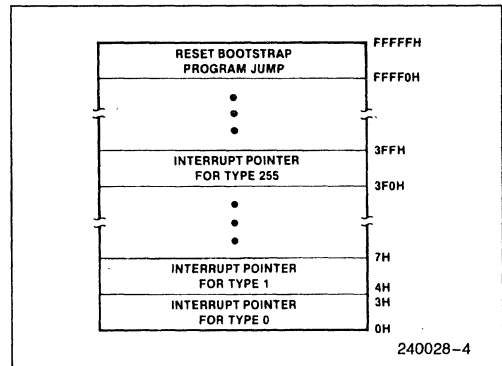


Figure 4. Reserved Memory Locations

Memory Reference Need	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch.
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
Local Data	DATA (DS)	Data references when: relative to stack, destination of string operation, or explicitly overridden.
External (Global) Data	EXTRA (ES)	Destination of string operations: Explicitly selected using a segment override.

initialization routine. Following RESET, the CPU will always begin execution at location FFFF0H where the jump must be located. Locations 00000H through 003FFH are reserved for interrupt operations. Four-byte pointers consisting of a 16-bit segment address and a 16-bit offset address direct program flow to one of the 256 possible interrupt service routines. The pointer elements are assumed to have been stored at their respective places in reserved memory prior to the occurrence of interrupts.

MINIMUM AND MAXIMUM MODES

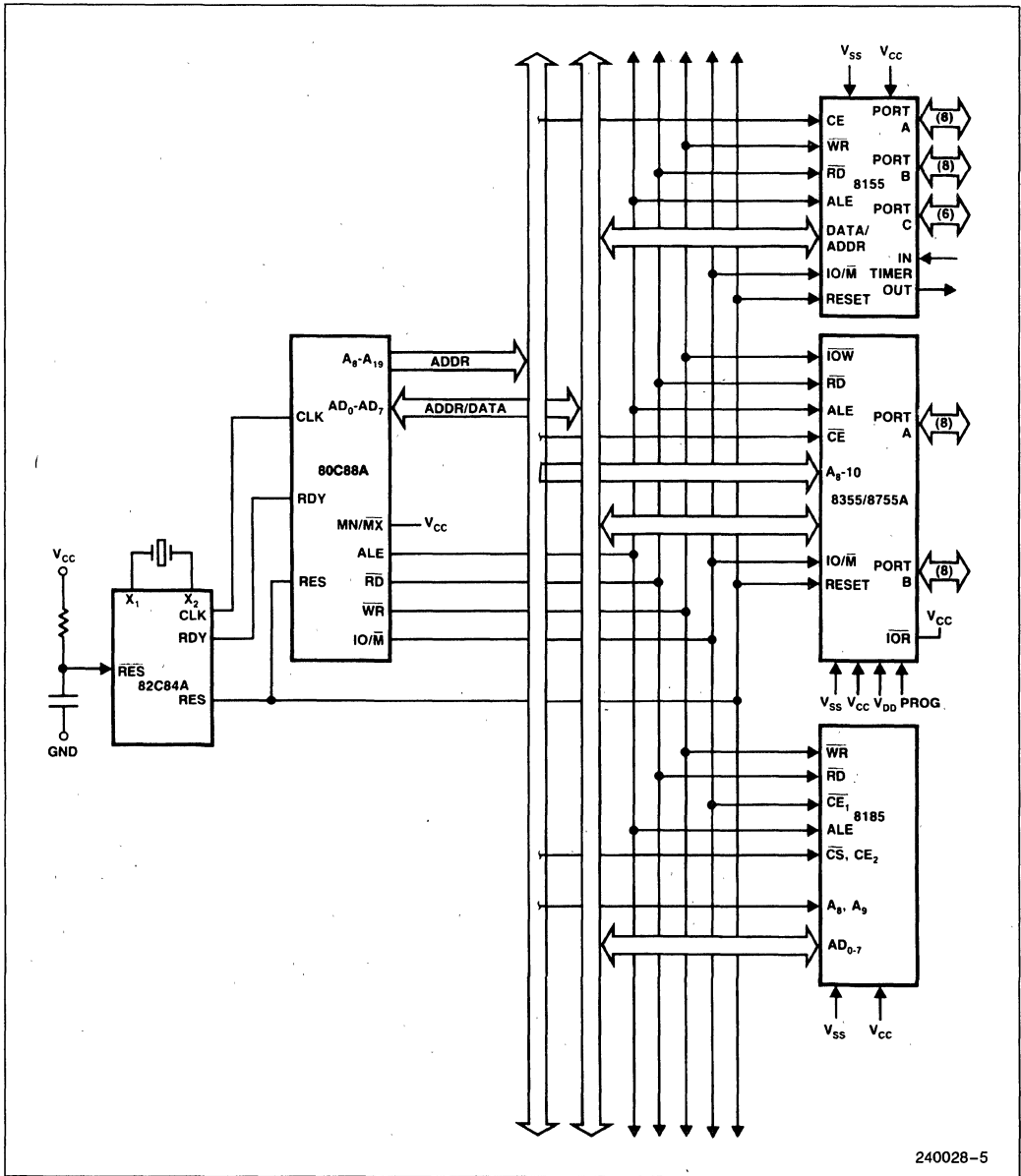
The requirements for supporting minimum and maximum 80C88A systems are sufficiently different that they cannot be done efficiently with 40 uniquely defined pins. Consequently, the 80C88A is equipped with a strap pin (MN/M \bar{X}) which defines the system configuration. The definition of a certain subset of the pins changes, dependent on the condition of the strap pin. When the MN/M \bar{X} pin is strapped to GND, the 80C88A defines pins 24 through 31 and 34 in maximum mode. When the MN/M \bar{X} pin is strapped to V_{CC}, the 80C88A generates bus control signals itself on pins 24 through 31 and 34.

The minimum mode 80C88A can be used with either a multiplexed or demultiplexed bus. The multiplexed bus configuration is compatible with the MCS[®]-85

multiplexed bus peripherals (8155, 8156, 8355, 8755A, and 8185). This configuration (See Figure 5) provides the user with a minimum chip count system. This architecture provides the 80C88A processing power in a highly integrated form.

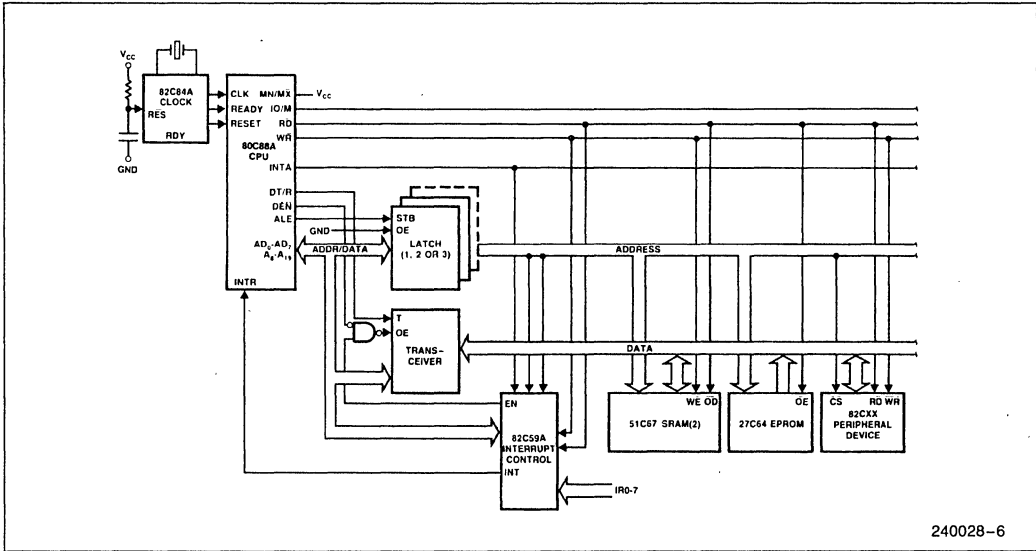
The demultiplexed mode requires one latch (for 64k addressability) or two latches (for a full megabyte of addressing). A third latch can be used for buffering if the address bus loading requires it. A transceiver can also be used if data bus buffering is required. (See Figure 6.) The 80C88A provides \overline{DEN} and DT/ \overline{R} to control the transceiver, and ALE to latch the addresses. This configuration of the minimum mode provides the standard demultiplexed bus structure with heavy bus buffering and relaxed bus timing requirements.

The maximum mode employs the 82C88 bus controller. (See Figure 7.) The 82C88 decodes status lines $\overline{S0}$, $\overline{S1}$, and $\overline{S2}$, and provides the system with all bus control signals. Moving the bus control to the 82C88 provides better source and sink current capability to the control lines, and frees the 80C88A pins for extended large system features. Hardware lock, queue status, and two request/grant interfaces are provided by the 80C88A in maximum mode. These features allow co-processors in local bus and remote bus configurations.



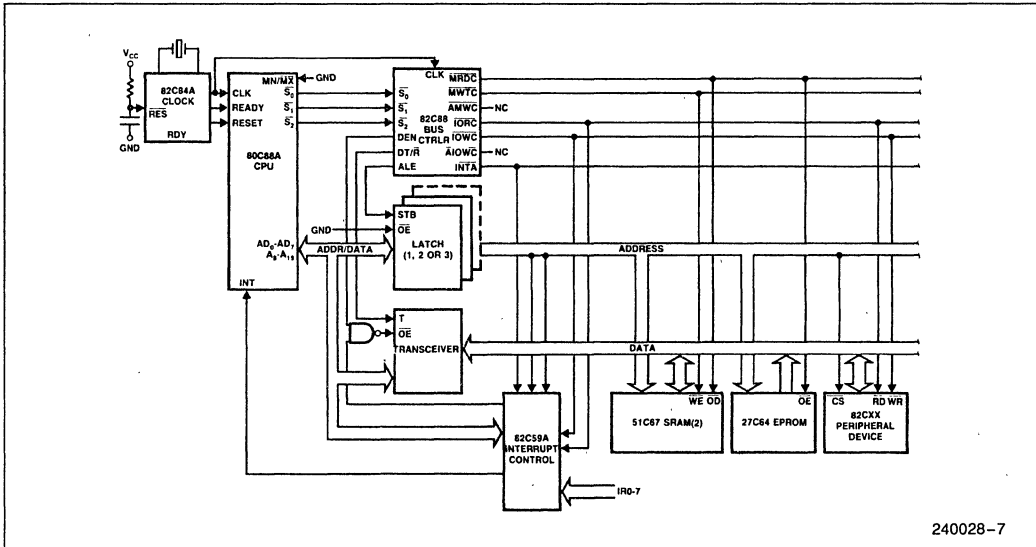
240028-5

Figure 5. Multiplexed Bus Configuration



240028-6

Figure 6. Demultiplexed Bus Configuration



240028-7

Figure 7. Fully Buffered System Using Bus Controller

Bus Operation

The 80C88A address/data bus is broken into three parts—the lower eight address/data bits (AD0–AD7), the middle eight address bits (A8–A15), and the upper four address bits (A16–A19). The address/data bits and the highest four address bits are time multiplexed. This technique provides the most efficient use of pins on the processor. The middle eight address bits are not multiplexed, i.e. they remain valid throughout each bus cycle. In addition, the bus can be demultiplexed at the processor with a single address latch if a standard, non-multiplexed bus is desired for the system.

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T1, T2, T3, and T4. (See Figure 8). The address is emitted from the processor during T1 and data transfer occurs on the bus during T3 and T4. T2 is used primarily for changing the direction of the bus during read operations. In the event that a “NOT READY” indication is given by the addressed device, “wait” states (Tw) are inserted between T3 and T4. Each inserted “wait” state is of the same duration as a CLK cycle. Periods can occur between 80C88A driven bus cycles. These are referred to as “idle” states (Ti), or inactive CLK cycles. The processor uses these cycles for internal housekeeping.

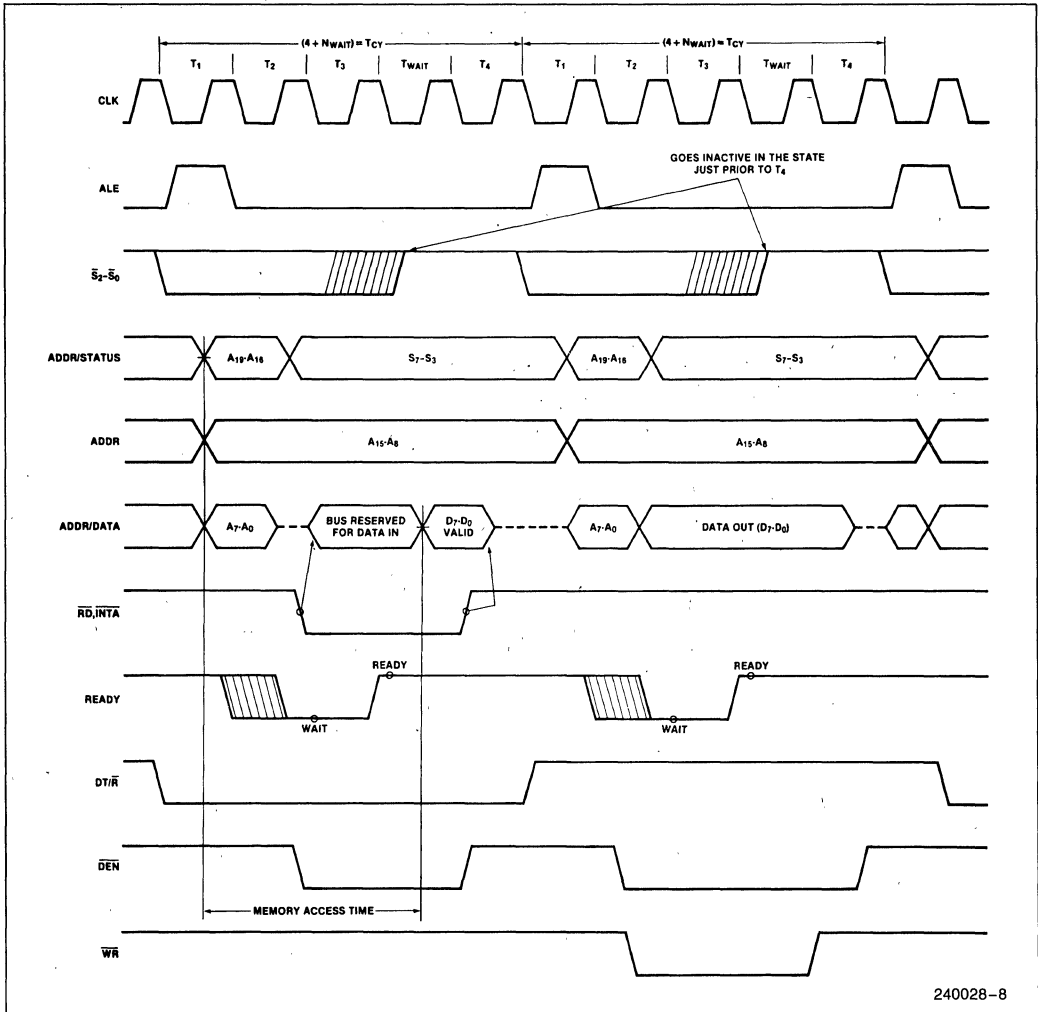


Figure 8. Basic System Timing

240028-8

During T1 of any bus cycle, the ALE (address latch enable) signal is emitted (by either the processor or the 82C88 bus controller, depending on the MN/MX strap). At the trailing edge of this pulse, a valid address and certain status information for the cycle may be latched.

Status bits $\overline{S_0}$, $\overline{S_1}$, and $\overline{S_2}$ are used by the bus controller, in maximum mode, to identify the type of bus transaction according to the following table:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CHARACTERISTICS
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

Status bits S3 through S6 are multiplexed with high order address bits and are therefore valid during T2 through T4. S3 and S4 indicate which segment register was used for this bus cycle in forming the address according to the following table:

S4	S3	CHARACTERISTICS
0 (LOW)	0	Alternate Data (extra segment)
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data

S5 is a reflection of the PSW interrupt enable bit. S6 is equal to 0.

I/O ADDRESSING

In the 80C88A, I/O operations can address up to a maximum of 64k I/O registers. The I/O address appears in the same format as the memory address on bus lines A15-A0. The address lines A19-A16 are zero in I/O operations. The variable I/O instructions, which use register DX as a pointer, have full address

capability, while the direct I/O instructions directly address one or two of the 256 I/O byte locations in page 0 of the I/O address space. I/O ports are addressed in the same manner as memory locations.

Designers familiar with the 8085 or upgrading an 8085 design should note that the 8085 addresses I/O with an 8-bit address on both halves of the 16-bit address bus. The 80C88A uses a full 16-bit address on its lower 16 address lines.

EXTERNAL INTERFACE

PROCESSOR RESET AND INITIALIZATION

Processor initialization or start up is accomplished with activation (HIGH) of the RESET pin. The 80C88A RESET is required to be HIGH for four or more clock cycles. The 80C88A will terminate operations on the high-going edge of RESET and will remain dormant as long as RESET is HIGH. The low-going transition of RESET triggers an internal reset sequence for approximately 7 clock cycles. After this interval the 80C88A operates normally, beginning with the instruction in absolute location FFFF0H. (See Figure 4.) The RESET input is internally synchronized to the processor clock. At initialization, the HIGH to LOW transition of RESET must occur no sooner than 50 μ s after power up, to allow complete initialization of the 80C88A.

NMI asserted prior to the 2nd clock after the end of RESET will not be honored. If NMI is asserted after that point and during the internal reset sequence, the processor may execute one instruction before responding to the interrupt. A hold request active immediately after RESET will be honored before the first instruction fetch.

All 3-state outputs float to 3-state OFF⁽¹⁾ during RESET. Status is active in the idle state for the first clock after RESET becomes active and then floats to 3-state OFF⁽¹⁾. ALE and HLDA are driven low.

NOTE:

1. See the section on Bus Hold Circuitry.

BUS HOLD CIRCUITRY

To avoid high current conditions caused by floating inputs to CMOS devices and to eliminate the need for pull-up/down resistors, "bus-hold" circuitry has been used on the 80C88A pins 2-16, 26-32, and 34-39 (Figure 9a, 9b). These circuits will maintain the last valid logic state if no driving source is present (i.e. an unconnected pin or a driving source which goes to a high impedance state). To overdrive the "bus hold" circuits, an external driver must be capable of supplying 350 μ A minimum sink or source current at valid input voltage levels. Since this "bus hold" circuitry is active and not a "resistive" type element, the associated power supply

current is negligible and power dissipation is significantly reduced when compared to the use of passive pull-up resistors.

INTERRUPT OPERATIONS

Interrupt operations fall into two classes: software or hardware initiated. The software initiated interrupts and software aspects of hardware interrupts are specified in the instruction set description in the iAPX 88 book or the iAPX 86,88 User's Manual. Hardware interrupts can be classified as nonmaskable or maskable.

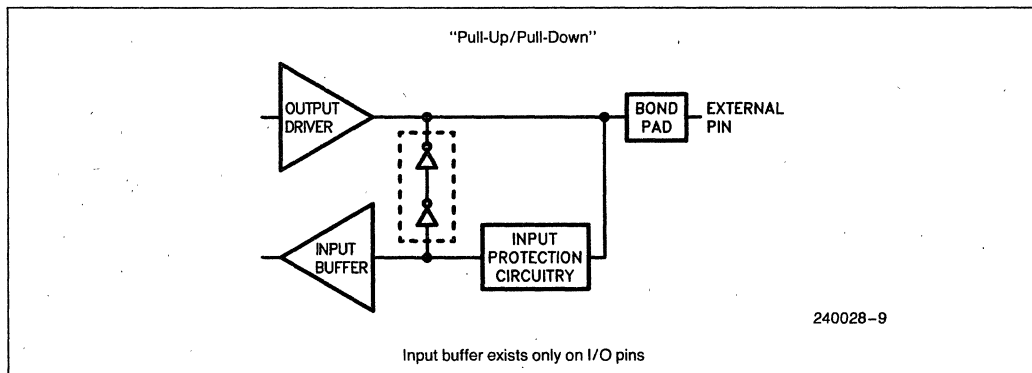


Figure 9a. Bus hold circuitry pin 2-16, 35-39.

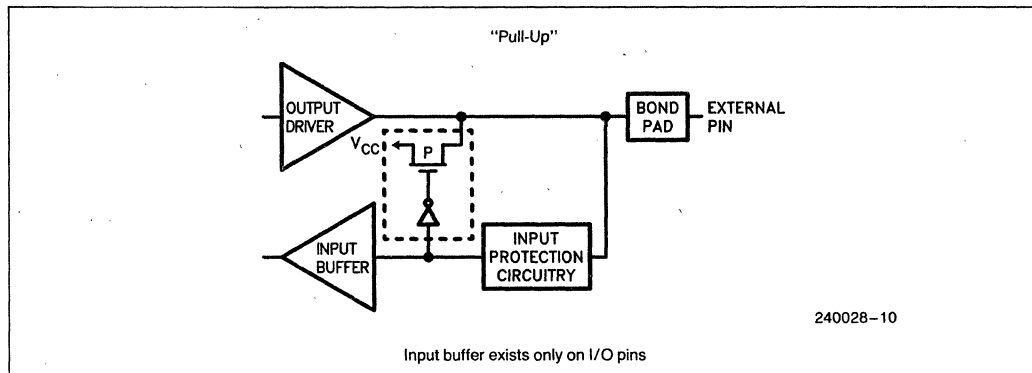


Figure 9b. Bus hold circuitry pin 26-32, 34.

Interrupts result in a transfer of control to a new program location. A 256 element table containing address pointers to the interrupt service program locations resides in absolute locations 0 through 3FFH (See Figure 4), which are reserved for this purpose. Each element in the table is 4 bytes in size and corresponds to an interrupt "type." An interrupting device supplies an 8-bit type number, during the interrupt acknowledge sequence, which is used to vector through the appropriate element to the new interrupt service program location.

NON-MASKABLE INTERRUPT (NMI)

The processor provides a single non-maskable interrupt (NMI) pin which has higher priority than the maskable interrupt request (INTR) pin. A typical use would be to activate a power failure routine. The NMI is edge-triggered on a LOW to HIGH transition. The activation of this pin causes a type 2 interrupt.

NMI is required to have a duration in the HIGH state of greater than two clock cycles, but is not required to be synchronized to the clock. Any higher going transition of NMI is latched on-chip and will be serviced at the end of the current instruction or between whole moves (2 bytes in the case of word moves) of a block type instruction. Worst case response to NMI would be for multiply, divide, and variable shift instructions. There is no specification on the occurrence of the low-going edge; it may occur before, during, or after the servicing of NMI. Another high-going edge triggers another response if it occurs after the start of the NMI procedure. The signal must

be free of logical spikes in general and be free of bounces on the low-going edge to avoid triggering extraneous responses.

MASKABLE INTERRUPT (INTR)

The 80C88A provides a single interrupt request input (INTR) which can be masked internally by software with the resetting of the interrupt enable (IF) flag bit. The interrupt request signal is level triggered. It is internally synchronized during each clock cycle on the high-going edge of CLK. To be responded to, INTR must be present (HIGH) during the clock period preceding the end of the current instruction or the end of a whole move for a block type instruction. During interrupt response sequence, further interrupts are disabled. The enable bit is reset as part of the response to any interrupt (INTR, NMI, software interrupt, or single step), although the FLAGS register which is automatically pushed onto the stack reflects the state of the processor prior to the interrupt. Until the old FLAGS register is restored, the enable bit will be zero unless specifically set by an instruction.

During the response sequence (See Figure 10), the processor executes two successive (back to back) interrupt acknowledge cycles. The 80C88A emits the LOCK signal (maximum mode only) from T2 of the first bus cycle until T2 of the second. A local bus "hold" request will not be honored until the end of the second bus cycle. In the second bus cycle, a

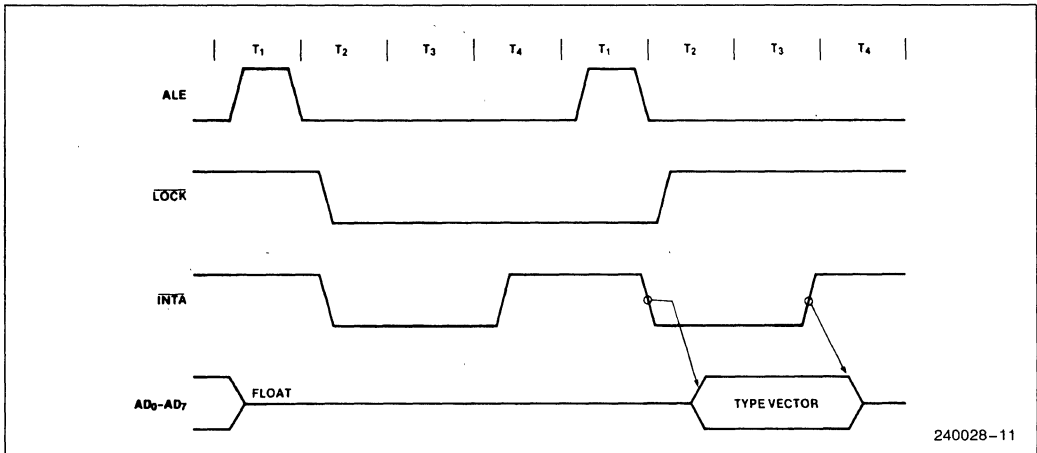


Figure 10. Interrupt Acknowledge Sequence

byte is fetched from the external interrupt system (e.g., 82C59A PIC) which identifies the source (type) of the interrupt. This byte is multiplied by four and used as a pointer into the interrupt vector lookup table. An INTR signal left HIGH will be continually responded to within the limitations of the enable bit and sample period. The interrupt return instruction includes a flags pop which returns the status of the original interrupt enable bit when it restores the flags.

HALT

When a software HALT instruction is executed, the processor indicates that it is entering the HALT state in one of two ways, depending upon which mode is strapped. In minimum mode, the processor issues ALE, delayed by one clock cycle, to allow the system to latch the halt status. Halt status is available on $\overline{IO/\overline{M}}$, $\overline{DT/\overline{R}}$, and $\overline{SS\overline{O}}$. In maximum mode, the processor issues appropriate HALT status on $\overline{S2}$, $\overline{S1}$, and $\overline{S0}$, and the 82C88 bus controller issues one ALE. The 80C88A will not leave the HALT state when a local bus hold is entered while in HALT. In this case, the processor reissues the HALT indicator at the end of the local bus hold. An interrupt request or RESET will force the 80C88A out of the HALT state.

READ/MODIFY/WRITE (SEMAPHORE) OPERATIONS VIA LOCK

The LOCK status information is provided by the processor when consecutive bus cycles are required during the execution of an instruction. This allows the processor to perform read/modify/write operations on memory (via the "exchange register with memory" instruction), without another system bus master receiving intervening memory cycles. This is useful in multiprocessor system configurations to accomplish "test and set lock" operations. The \overline{LOCK} signal is activated (LOW) in the clock cycle following decoding of the LOCK prefix instruction. It is deactivated at the end of the last bus cycle of the instruction following the LOCK prefix. While \overline{LOCK} is active, a request on a $\overline{RQ/\overline{GT}}$ pin will be recorded, and then honored at the end of the LOCK.

EXTERNAL SYNCHRONIZATION VIA \overline{TEST}

As an alternative to interrupts, the 80C88A provides a single software-testable input pin (\overline{TEST}). This input is utilized by executing a WAIT instruction. The single WAIT instruction is repeatedly executed until the \overline{TEST} input goes active (LOW). The execution of WAIT does not consume bus cycles once the queue is full.

If a local bus request occurs during WAIT execution, the 80C88A 3-states all output drivers. If interrupts are enabled, the 80C88A will recognize interrupts and process them. The WAIT instruction is then re-fetched, and reexecuted.

BASIC SYSTEM TIMING

In minimum mode, the $\overline{MN/\overline{MX}}$ pin is strapped to V_{CC} and the processor emits bus control signals compatible with the 8085 bus structure. In maximum mode, the $\overline{MN/\overline{MX}}$ pin is strapped to GND and the processor emits coded status information which the 82C88 bus controller uses to generate MULTIBUS compatible bus control signals.

System Timing — Minimum System

(See Figure 8.)

The read cycle begins in T1 with the assertion of the address latch enable (ALE) signal. The trailing (low going) edge of this signal is used to latch the address information, which is valid on the address/data bus (AD0-AD7) at this time, into a latch. Address lines A8 through A15 do not need to be latched because they remain valid throughout the bus cycle. From T1 to T4 the $\overline{IO/\overline{M}}$ signal indicates a memory or I/O operation. At T2 the address is removed from the address/data bus and the bus goes to a high impedance state. The read control signal is also asserted at T2. The read (\overline{RD}) signal causes the addressed device to enable its data bus drivers to the local bus. Some time later, valid data will be available on the bus and the addressed device will drive the READY line HIGH. When the processor returns the read signal to a HIGH level, the addressed device will again 3-state its bus drivers. If a transceiver is required to buffer the 80C88A local bus, signals $\overline{DT/\overline{R}}$ and \overline{DEN} are provided by the 80C88A.

A write cycle also begins with the assertion of ALE and the emission of the address. The $\overline{IO/\overline{M}}$ signal is again asserted to indicate a memory or I/O write operation. In T2, immediately following the address emission, the processor emits the data to be written into the addressed location. This data remains valid until at least the middle of T4. During T2, T3, and T_w , the processor asserts the write control signal. The write (\overline{WR}) signal becomes active at the beginning of T2, as opposed to the read, which is delayed somewhat into T2 to provide time for the bus to float.

The basic difference between the interrupt acknowledge cycle and a read cycle is that the interrupt acknowledge (\overline{INTA}) signal is asserted in place of the read (\overline{RD}) signal and the address bus is floated. (See Figure 10.) In the second of two successive \overline{INTA} cycles, a byte of information is read from the data bus, as supplied by the interrupt system logic (i.e. 82C59A priority interrupt controller). This byte identifies the source (type) of the interrupt. It is multiplied by four and used as a pointer into the interrupt vector lookup table, as described earlier.

BUS TIMING — MEDIUM COMPLEXITY SYSTEMS

(See Figure 11.)

For medium complexity systems, the MN/\overline{MX} pin is connected to GND and the 82C88 bus controller is added to the system, as well as a latch for latching the system address, and a transceiver to allow for bus loading greater than the 80C88A is capable of handling. Signals ALE , \overline{DEN} , and DT/\overline{R} are generated by the 82C88 instead of the processor in this configuration, although their timing remains relatively the same. The 80C88A status outputs (S_2 , S_1 , and S_0) provide type of cycle information and become 82C88 inputs. This bus cycle information specifies read (code, data, or I/O), write (data or I/O), interrupt acknowledge, or software halt. The 82C88 thus issues control signals specifying memory read or write, I/O read or write, or interrupt acknowledge. The 82C88 provides two types of write strobes, normal and advanced, to be applied as required. The normal write strobes have data valid at the leading edge of write. The advanced write strobes have the same timing as read strobes, and hence, data is not valid at the leading edge of write. The transceiver receives the usual T and \overline{OE} inputs from the 82C88's DT/\overline{R} and \overline{DEN} outputs.

The pointer into the interrupt vector table, which is passed during the second \overline{INTA} cycle, can derive from an 82C59A located on either the local bus or the system bus. If the master 82C59A priority interrupt controller is positioned on the local bus, a TTL gate is required to disable the transceiver when reading from the master 82C59A during the interrupt acknowledge sequence and software "poll".

THE 80C88A COMPARED TO THE 80C86

The 80C88A CPU is an 8-bit processor designed around the 80C86 internal structure. Most internal functions of the 80C88A are identical to the equivalent

80C86 functions. The 80C88A handles the external bus the same way the 80C86 does with the distinction of handling only 8 bits at a time. Sixteen-bit operands are fetched or written in two consecutive bus cycles. Both processors will appear identical to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions have the same end result. The differences between the 80C88A and 80C86 are outlined below. The engineer who is unfamiliar with the 80C86 is referred to the iAPX 86, 88 User's Manual, Chapters 2 and 4, for function description and instruction set information. Internally, there are three differences between the 80C88A and the 80C86. All changes are related to the 8-bit bus interface.

- The queue length is 4 bytes in the 80C88A, whereas the 80C86 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The 80C88A BIU will fetch a new instruction to load into the queue each time there is a 1 byte hole (space available) in the queue. The 80C86 waits until a 2-byte space is available.
- The internal execution time of the instruction set is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU is also limited by the speed of instruction fetches. This latter problem only occurs when a series of simple operations occur. When the more sophisticated instructions of the 80C88A are being used, the queue has time to fill and the execution proceeds as fast as the execution unit will allow.

The 80C88A and 80C86 are completely software compatible by virtue of their identical execution units. Software that is system dependent may not be completely transferable, but software that is not system dependent will operate equally as well on an 80C88A or an 80C86.

The hardware interface of the 80C88A contains the major differences between the two CPUs. The pin assignments are nearly identical, however with the following functional changes:

- A8–A15 — These pins are only address outputs on the 80C88A. These address lines are latched internally and remain valid throughout a bus cycle in a manner similar to the 8085 upper address lines.

- \overline{BHE} has no meaning on the 80C88A and has been eliminated.
- \overline{SSO} provides the $\overline{S0}$ status information in the minimum mode. This output occurs on pin 34 in minimum mode only. DT/\overline{R} , IO/\overline{M} , and \overline{SSO} provide the complete bus status in minimum mode.
- IO/\overline{M} has been inverted to be compatible with the MCS-85 bus structure.
- ALE is delayed by one clock cycle in the minimum mode when entering HALT, to allow the status to be latched with ALE.

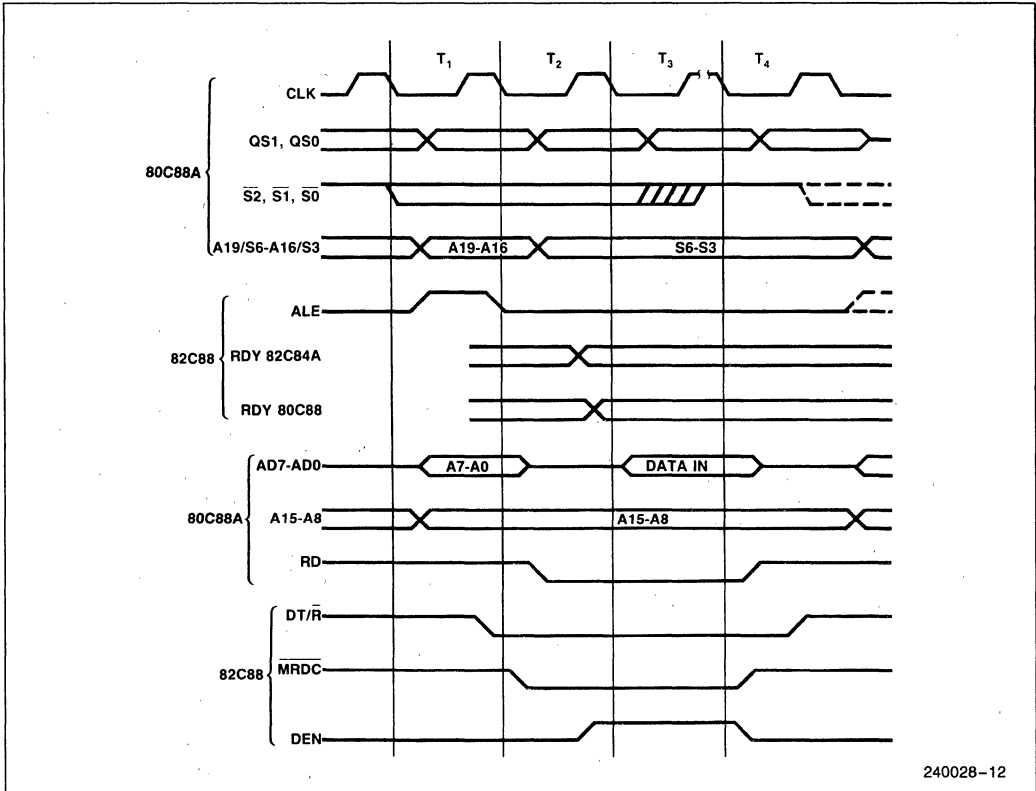


Figure 11. Medium Complexity System Timing

ABSOLUTE MAXIMUM RATINGS*

Supply Voltage (With respect to ground)	-0.5 to 7.0V
Input Voltage Applied (w.r.t. ground)	-0.5 to $V_{CC} + 0.5V$
Output Voltage Applied (w.r.t. ground)	-0.5 to $V_{CC} + 0.5V$
Power Dissipation	1.0W
Storage Temperature	-65°C to +150°C
Ambient Temperature Under Bias	0°C to +70°C

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $T_A = 0^\circ\text{C to } 70^\circ\text{C}$, $V_{CC} = 5V \pm 5\%$

Symbol	Parameter	80C88A-2		Units	Test Conditions
		Min	Max		
V_{IL}	Input Low Voltage	-0.5	+0.8	V	
V_{IH}	Input High Voltage (All inputs except clock)	2.0		V	
V_{CH}	Clock High Voltage	$V_{CC} - 0.8$		V	
V_{OL}	Output Low Voltage		0.45	V	$I_{OL} = 2.5 \text{ mA}$
V_{OH}	Output High Voltage	3.0 $V_{CC} - 0.4$		V	$I_{OH} = -2.5 \text{ mA}$ $I_{OH} = -100 \mu\text{A}$
I_{CC}	Power Supply Current		10 mA/MHz		$V_{IL} = \text{GND}$, $V_{IH} = V_{CC}$
I_{CCS}	Standby Supply Current		500	μA	$V_{IN} = V_{CC}$ or GND Outputs Unloaded CLK = GND or V_{CC}
I_{LI}	Input Leakage Current		± 1.0	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{BHL}	Input Leakage Current (Bus Hold Low)	50	400	μA	$V_{IN} = 0.8V$ (Note 4)
I_{BHH}	Input Leakage Current (Bus Hold High)	-50	-400	μA	$V_{IN} = 3.0V$ (Note 5)
I_{BHLO}	Bus Hold Low Overdrive		600	μA	(Note 2)
I_{BHHO}	Bus Hold High Overdrive		-600	μA	(Note 3)
I_{LO}	Output Leakage Current		± 10	μA	$V_{OUT} = \text{GND}$ or V_{CC}
C_{IN}	Capacitance of Input Buffer (All inputs except AD_0 - AD_7 , $\overline{RQ}/\overline{GT}$)		5	pF	(Note 1)
C_{IO}	Capacitance of I/O Buffer (AD_0 - AD_7 , $\overline{RQ}/\overline{GT}$)		20	pF	(Note 1)
C_{OUT}	Output Capacitance		15	pF	(Note 1)

NOTES:

1. Characterization conditions are a) Frequency = 1 MHz, b) Unmeasured pins at GND
c) V_{IN} at +5.0V or GND.
2. An external driver must source at least I_{BHLO} to switch this node from LOW to HIGH.
3. An external driver must sink at least I_{BHHO} to switch this node from HIGH to LOW.
4. Test condition is to lower V_{IN} to GND and then raise V_{IN} to 0.8V on pins 2-16 and 34-39.
5. Test condition is to raise V_{IN} to V_{CC} and then lower V_{IN} to 3.0V on pins 2-16, 26-32 and 34-39.

A.C. CHARACTERISTICS $T_A = 0^\circ\text{C to } 70^\circ\text{C}$, $V_{CC} = 5\text{V} \pm 5\%$
MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

Symbol	Parameter	80C88A-2		Units	Test Conditions	
		Min	Max			
TCLCL	CLK Cycle Period	125	D.C.	ns		
TCLCH	CLK Low Time	68		ns		
TCHCL	CLK High Time	44		ns		
TCH1CH2	CLK Rise Time		10	ns	From 1.0V to 3.5V	
TCL2CL1	CLK Fall Time		10	ns	From 3.5V to 1.0V	
TDVCL	Data in Setup Time	20		ns		
TCLDX	Data in Hold Time	10		ns		
TR1VCL	RDY Setup Time into 82C84A (Notes 1, 2)	35		ns		
TCLR1X	RDY Hold Time into 82C84A (Notes 1, 2)	0		ns		
TRYHCH	READY Setup Time into 80C88A	68		ns		
TCHRYX	READY Hold Time into 80C88A	20		ns		
TRYLCL	READY Inactive to CLK (Note 3)	-8		ns		
THVCH	HOLD Setup Time	20		ns		
TINVCH	INTR, NMI, TEST Setup Time (Note 2)	15		ns		
TILIH	Input Rise Time (Except CLK) (Note 4)		15	ns		From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK) (Note 4)		15	ns		From 2.0V to 0.8V

A.C. CHARACTERISTICS (Continued)

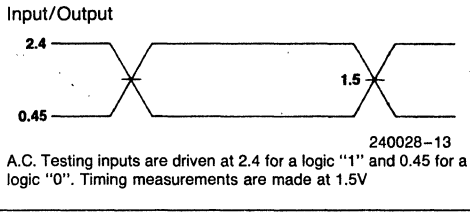
TIMING RESPONSES

Symbol	Parameter	80C88A-2		Units	Test Conditions
		Min	Max		
TCLAV	Address Valid Delay	10	60	ns	
TCLAX	Address Hold Time	10		ns	
TCLAZ	Address Float Delay	TCLAX	50	ns	
TLHLL	ALE Width	TCLCH-10		ns	
TCLLH	ALE Active Delay		50	ns	
TCHLL	ALE Inactive Delay		55	ns	
TLLAX	Address Hold Time to ALE Inactive	TCHCL-10		ns	
TCLDV	Data Valid Delay	10	60	ns	
TCHDX	Data Hold Time	10		ns	
TWHDX	Data Hold Time After \overline{WR}	TCLCH-30		ns	
TCVCTV	Control Active Delay 1	10	70	ns	
TCHCTV	Control Active Delay 2	10	60	ns	
TCVCTX	Control Inactive Delay	10	70	ns	
TAZRL	Address Float to READ Active	0		ns	
TCLRL	\overline{RD} Active Delay	10	100	ns	
TCLRHR	\overline{RD} Inactive Delay	10	80	ns	
TRHAV	\overline{RD} Inactive to Next Address Active	TCLCL-40		ns	
TCLHAV	HLDA Valid Delay	10	100	ns	
TRLRH	\overline{RD} Width	2TCLCL-50		ns	
TWLWH	\overline{WR} Width	2TCLCL-40		ns	
TAVAL	Address Valid to ALE Low	TCLCH-40		ns	
TOLOH	Output Rise Time (Note 4)		15	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time (Note 4)		15	ns	From 2.0V to 0.8V

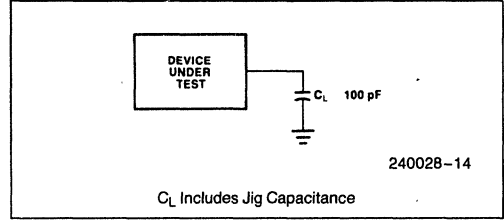
NOTES:

- Signal at 82C84A shown for reference only. See 82C84A data sheet for the most recent specifications.
- Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
- Applies only to T2 state (8 ns into T3 state).
- These parameters are characterized and not 100% tested.

A.C. TESTING INPUT, OUTPUT WAVEFORM

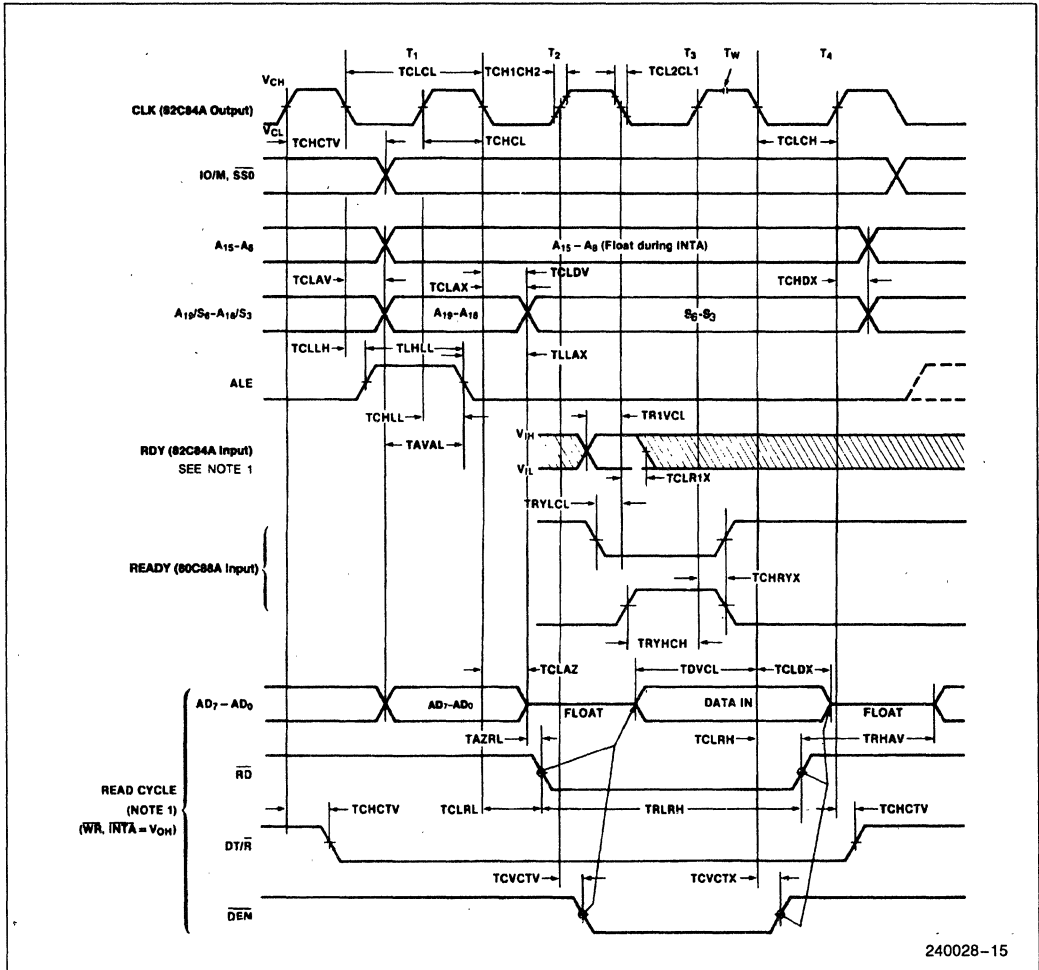


A.C. TESTING LOAD CIRCUIT



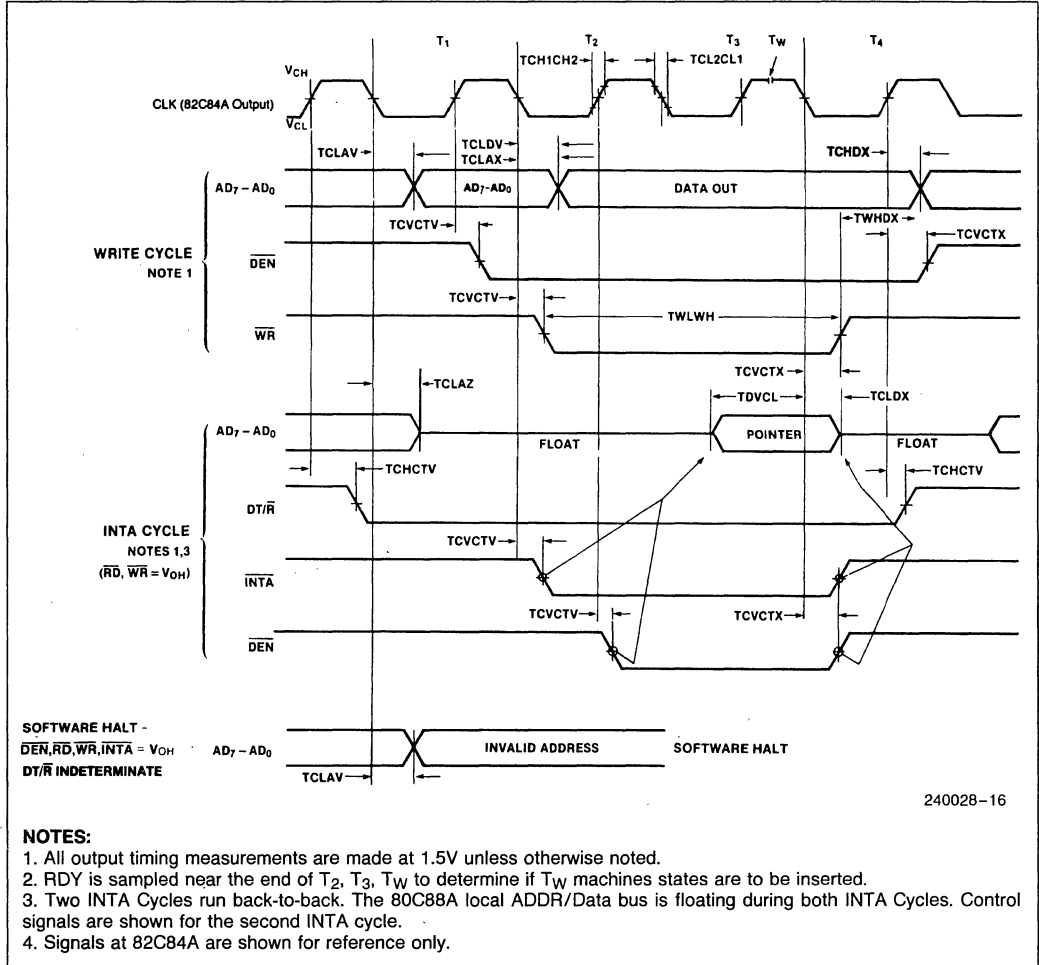
WAVEFORMS

BUS TIMING — MINIMUM MODE SYSTEM



WAVEFORMS (Continued)

BUS TIMING — MINIMUM MODE SYSTEM (Continued)



NOTES:

- All output timing measurements are made at 1.5V unless otherwise noted.
- RDY is sampled near the end of T₂, T₃, T_W to determine if T_W machines states are to be inserted.
- Two INTA Cycles run back-to-back. The 80C88A local ADDR/Data bus is floating during both INTA Cycles. Control signals are shown for the second INTA cycle.
- Signals at 82C84A are shown for reference only.

A.C. CHARACTERISTICS
**MAX MODE SYSTEM (USING 82C88 BUS CONTROLLER)
TIMING REQUIREMENTS**

Symbol	Parameter	80C88A-2		Units	Test Conditions	
		Min	Max			
TCLCL	CLK Cycle Period	125	D.C.	ns		
TCLCH	CLK Low Time	68		ns		
TCHCL	CLK High Time	44		ns		
TCH1CH2	CLK Rise Time		10	ns	From 1.0V to 3.5V	
TCL2CL1	CLK Fall Time		10	ns	From 3.5V to 1.0V	
TDVCL	Data In Setup Time	20		ns		
TCLDX	Data In Hold Time	10		ns		
TR1VCL	RDY Setup Time into 82C84 (See Notes 1, 2)	35		ns		
TCLR1X	RDY Hold Time into 82C84 (See Notes 1, 2)	0		ns		
TRYHCH	READY Setup Time into 80C88A	68		ns		
TCHRYX	READY Hold Time into 80C88A	20		ns		
TRYLCL	READY Inactive to CLK (See Note 4)	-8		ns		
TINVCH	Setup Time for Recognition (INTR, NMI, TEST) (See Note 2)	15		ns		
TGVCH	RQ/GT Setup Time	15		ns		
TCHGX	RQ Hold Time into 80C88A	30		ns		
TILIH	Input Rise Time (Except CLK) (Note 5)		15	ns		From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK) (Note 5)		15	ns		From 2.0V to 0.8V

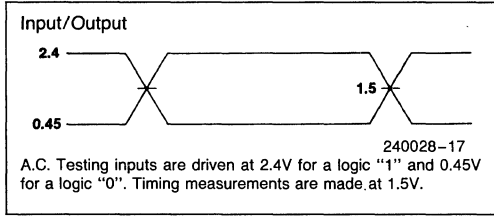
A.C. CHARACTERISTICS
TIMING RESPONSES

Symbol	Parameter	80C88A-2		Units	Test Conditions
		Min	Max		
TCLML	Command Active Delay (Note 1)	5	35	ns	
TCLMH	Command Inactive Delay (Note 1)	5	35	ns	
TRYHSH	READY Active to Status Passive (Note 3)		65	ns	
TCHSV	Status Active Delay	10	60	ns	
TCLSH	Status Inactive Delay	10	70	ns	
TCLAV	Address Valid Delay	10	60	ns	
TCLAX	Address Hold Time	10		ns	
TCLAZ	Address Float Delay	TCLAX	50	ns	
TSVLH	Status Valid to ALE High (Note 1)		20	ns	
TSMVCH	Status Valid to MCE High (Note 1)		30	ns	
TCLLH	CLK Low to ALE Valid (Note 1)		20	ns	
TCLMCH	CLK Low to MCE High (Note 1)		25	ns	
TCHLL	ALE Inactive Delay (Note 1)	4	18	ns	
TCLDV	Data Valid Delay	10	60	ns	
TCHDX	Data Hold Time	10		ns	
TCVNV	Control Active Delay (Note 1)	5	45	ns	
TCVNX	Control Inactive Delay (Note 1)	10	45	ns	
TAZRL	Address Float to Read Active	0		ns	
TCLRL	RD Active Delay	10	100	ns	
TCLRH	RD Inactive Delay	10	80	ns	
TRHAV	RD Inactive to Next Address Active	TCLCL-40		ns	
TCHDTL	Direction Control Active Delay (Note 1)		50	ns	
TCHDTH	Direction Control Inactive Delay (Note 1)		30	ns	
TCLGL	GT Active Delay	0	50	ns	
TCLGH	GT Inactive Delay	0	50	ns	
TRLRH	RD Width	2TCLCL-50		ns	
TOLOH	Output Rise Time (Note 5)		15	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time (Note 5)		15	ns	From 2.0V to 0.8V

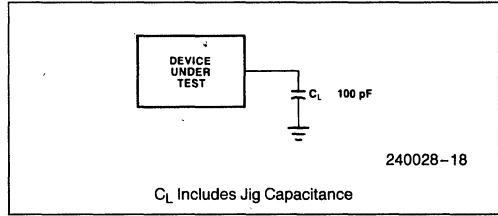
NOTES:

- Signal at 82C84A or 82C88 shown for reference only. See 82C84A and 82C88 data sheets for the most recent specifications.
- Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
- Applies only to T3 and wait states (8 ns into T3 state).
- Applies only to T2 state (8 ns into T3 state).
- These parameters are characterized and not 100% tested.

A.C. TESTING INPUT, OUTPUT WAVEFORM

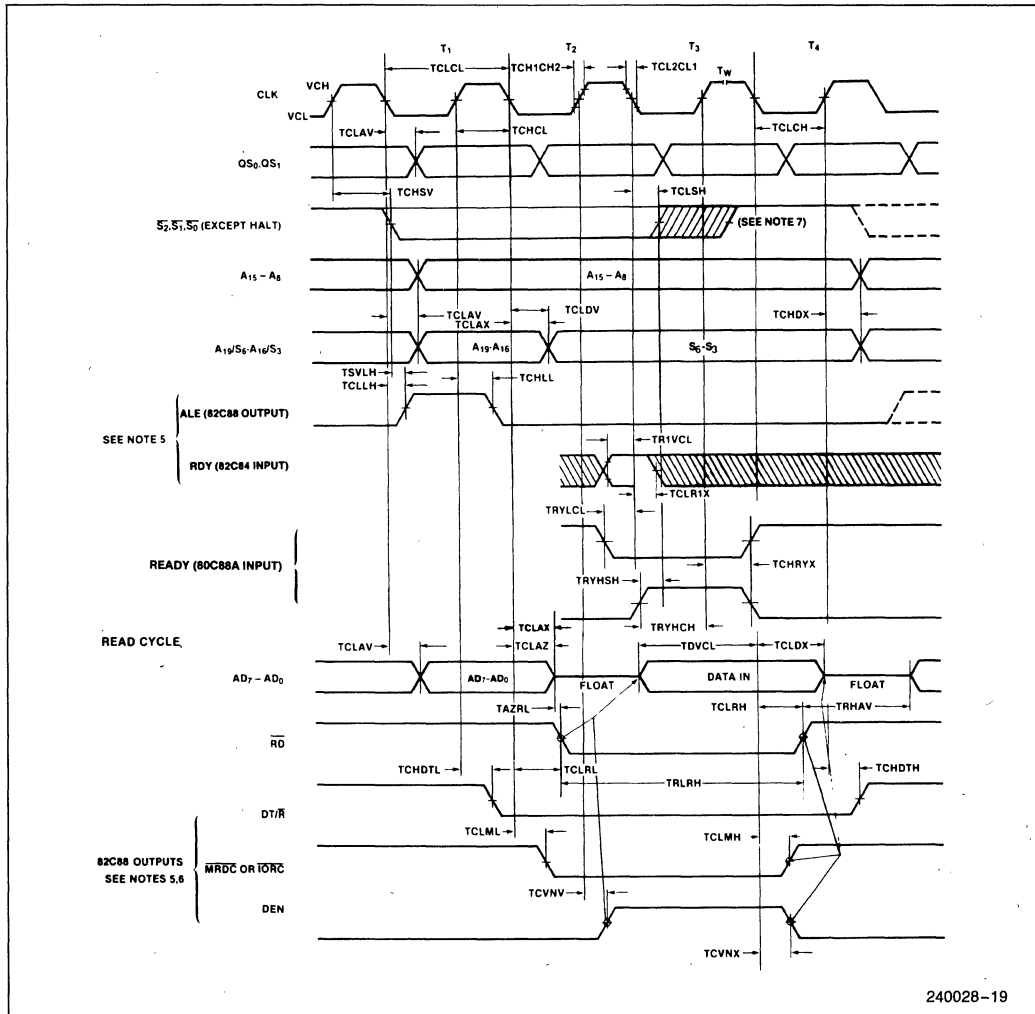


A.C. TESTING LOAD CIRCUIT

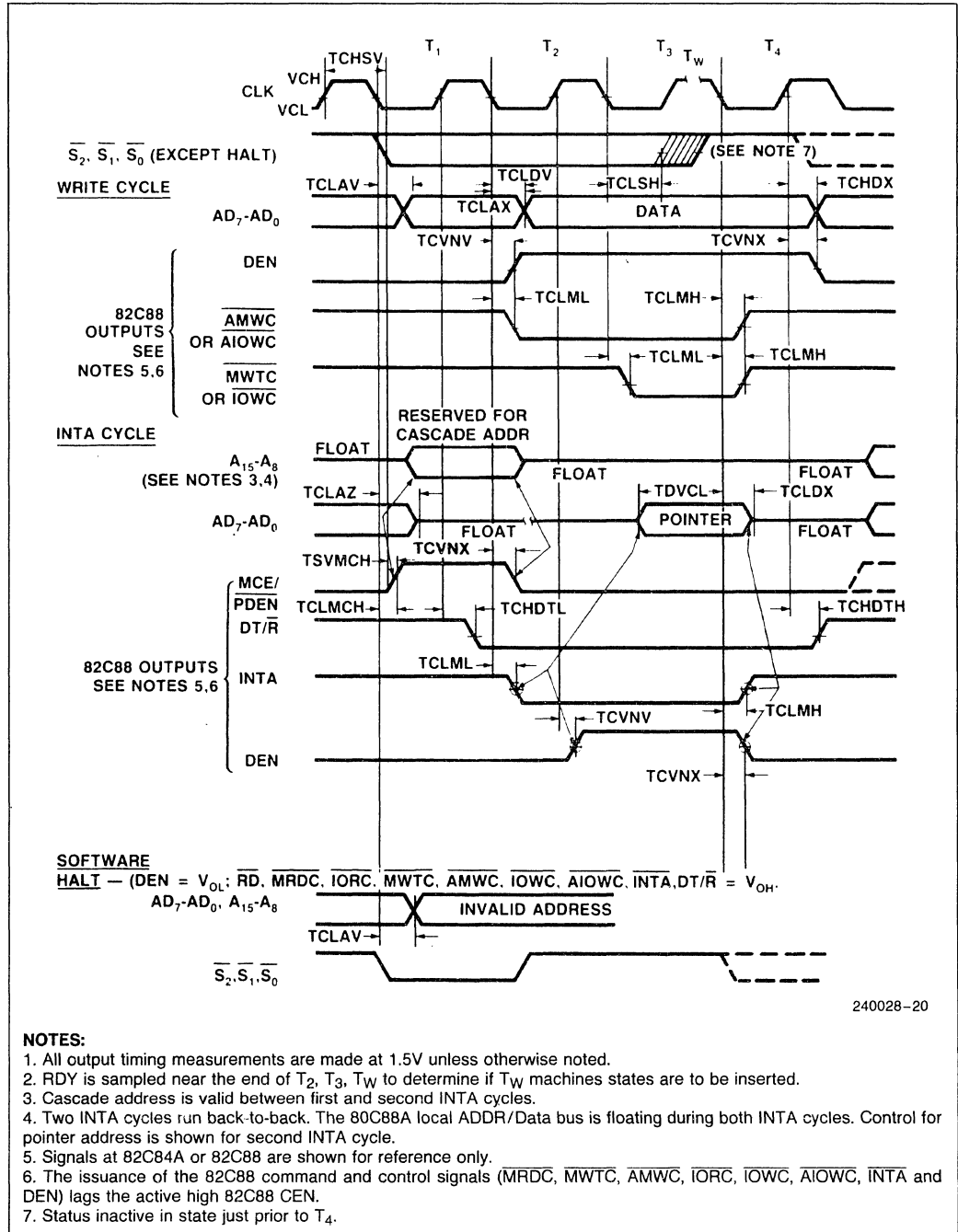


WAVEFORMS

BUS TIMING—MAXIMUM MODE



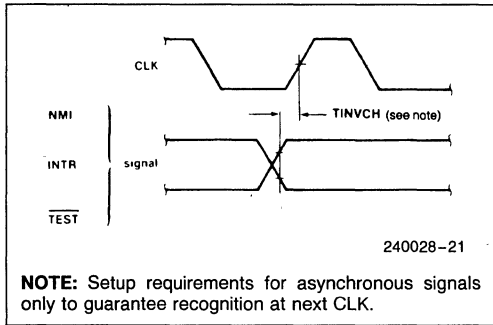
WAVEFORMS (Continued)

BUS TIMING — MAXIMUM MODE SYSTEM (USING 82C88)


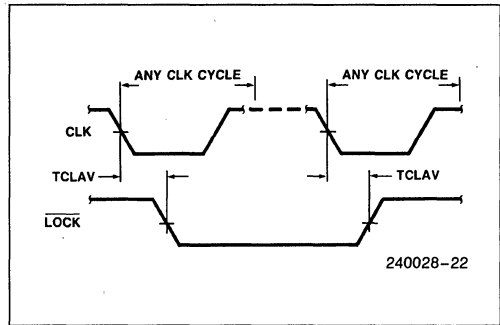
240028-20

WAVEFORMS (Continued)

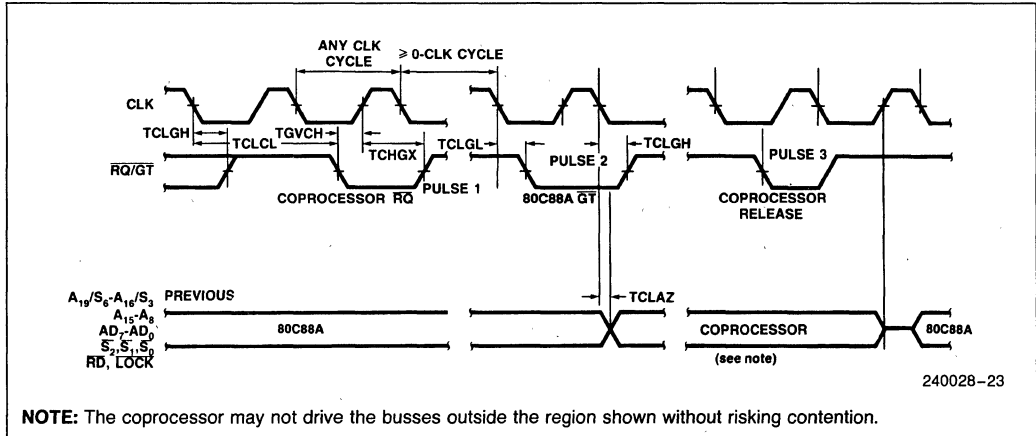
ASYNCHRONOUS SIGNAL RECOGNITION



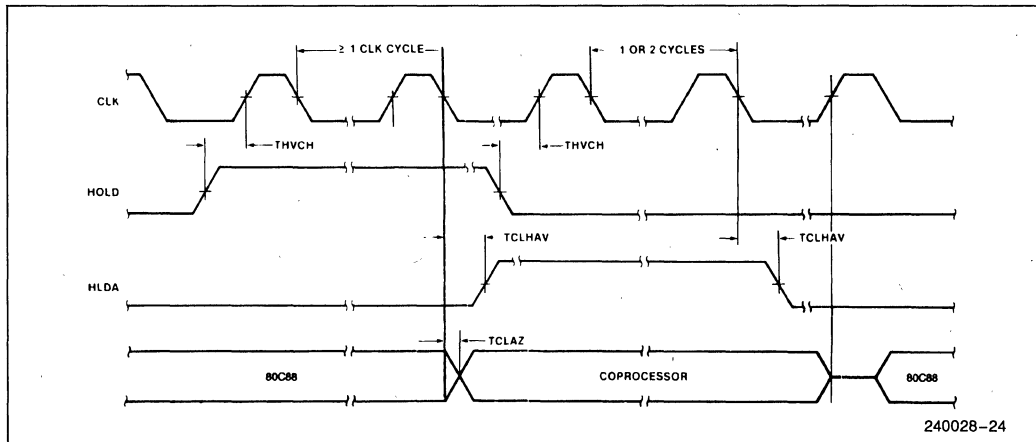
BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)



REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)



HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)



80C86A/80C88A INSTRUCTION SET SUMMARY

Mnemonic and Description	Instruction Code			
DATA TRANSFER				
MOV = Move:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Register/Memory to/from Register	1 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w 1
Immediate to Register	1 0 1 1 w reg	data	data if w 1	
Memory to Accumulator	1 0 1 0 0 0 0 w	add-low	addr-high	
Accumulator to Memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/Memory to Segment Register**	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment Register to Register/Memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		
PUSH = Push:				
Register/Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m		
Register	0 1 0 1 0 reg			
Segment Register	0 0 0 reg 1 1 0			
POP = Pop:				
Register/Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m		
Register	0 1 0 1 1 reg			
Segment Register	0 0 0 reg 1 1 1			
XCHG = Exchange:				
Register/Memory with Register	1 0 0 0 0 1 1 w	mod reg r/m		
Register with Accumulator	1 0 0 1 0 reg			
IN = Input from:				
Fixed Port	1 1 1 0 0 1 0 w	port		
Variable Port	1 1 1 0 1 1 0 w			
OUT = Output to:				
Fixed Port	1 1 1 0 0 1 1 w	port		
Variable Port	1 1 1 0 1 1 1 w			
XLAT = Translate Byte to AL	1 1 0 1 0 1 1 1			
LEA = Load EA to Register	1 0 0 0 1 1 0 1	mod reg r/m		
LDS = Load Pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m		
LES = Load Pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m		
LAHF = Load AH with Flags	1 0 0 1 1 1 1 1			
SAHF = Store AH into Flags	1 0 0 1 1 1 1 0			
PUSHF = Push Flags	1 0 0 1 1 1 0 0			
POPF = Pop Flags	1 0 0 1 1 1 0 1			

80C86A/80C88A INSTRUCTION SET SUMMARY (Continued)

Mnemonic and Description	Instruction Code			
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
ARITHMETIC				
ADD = Add:				
Reg./Memory with Register to Either	0 0 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s:w = 01
Immediate to Accumulator	0 0 0 0 0 1 0 w	data	data if w = 1	
ADC = Add with Carry:				
Reg./Memory with Register to Either	0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s:w = 01
Immediate to Accumulator	0 0 0 1 0 1 0 w	data	data if w = 1	
INC = Increment:				
Register/Memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m		
Register	0 1 0 0 0 reg			
AAA = ASCII Adjust for Add	0 0 1 1 0 1 1 1			
DAA = Decimal Adjust for Add	0 0 1 0 0 1 1 1			
SUB = Subtract:				
Reg./Memory and Register to Either	0 0 1 0 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s:w = 01
Immediate from Accumulator	0 0 1 0 1 1 0 w	data	data if w = 1	
SBB = Subtract with Borrow				
Reg./Memory and Register to Either	0 0 0 1 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s:w = 01
Immediate from Accumulator	0 0 0 1 1 1 0 w	data	data if w = 1	
DEC = Decrement:				
Register/Memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m		
Register	0 1 0 0 1 reg			
NEG = Change Sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		
CMP = Compare:				
Register/Memory and Register	0 0 1 1 1 0 d w	mod reg r/m		
Immediate with Register/Memory	1 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s:w = 01
Immediate with Accumulator	0 0 1 1 1 1 0 w	data	data if w = 1	
AAS = ASCII Adjust for Subtract	0 0 1 1 1 1 1 1			
DAS = Decimal Adjust for Subtract	0 0 1 0 1 1 1 1			
MUL = Multiply (Unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m		
IMUL = Integer Multiply (Signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m		
AAM = ASCII Adjust for Multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		
DIV = Divide (Unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m		
IDIV = Integer Divide (Signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m		
AAD = ASCII Adjust for Divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		
CBW = Convert Byte to Word	1 0 0 1 1 0 0 0			
CWD = Convert Word to Double Word	1 0 0 1 1 0 0 1			

80C86A/80C88A INSTRUCTION SET SUMMARY (Continued)

Mnemonic and Description	Instruction Code			
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
LOGIC				
NOT = Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m		
SHL/SAL = Shift Logical/Arithmetic Left	1 1 0 1 0 0 v w	mod 1 0 0 r/m		
SHR = Shift Logical Right	1 1 0 1 0 0 v w	mod 1 0 1 r/m		
SAR = Shift Arithmetic Right	1 1 0 1 0 0 v w	mod 1 1 1 r/m		
ROL = Rotate Left	1 1 0 1 0 0 v w	mod 0 0 0 r/m		
ROR = Rotate Right	1 1 0 1 0 0 v w	mod 0 0 1 r/m		
RCL = Rotate Through Carry Flag Left	1 1 0 1 0 0 v w	mod 0 1 0 r/m		
RCR = Rotate Through Carry Right	1 1 0 1 0 0 v w	mod 0 1 1 r/m		
AND = And:				
Reg./Memory and Register to Either	0 0 1 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w = 1
Immediate to Accumulator	0 0 1 0 0 1 0 w	data	data if w = 1	
TEST = And Function to Flags, No Result:				
Register/Memory and Register	1 0 0 0 0 1 0 w	mod reg r/m		
Immediate Data and Register/Memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate Data and Accumulator	1 0 1 0 1 0 0 w	data	data if w = 1	
OR = Or:				
Reg./Memory and Register to Either	0 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1
Immediate to Accumulator	0 0 0 0 1 1 0 w	data	data if w = 1	
XOR = Exclusive or:				
Reg./Memory and Register to Either	0 0 1 1 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1
Immediate to Accumulator	0 0 1 1 0 1 0 w	data	data if w = 1	
STRING MANIPULATION				
REP = Repeat	1 1 1 1 0 0 1 z			
MOVS = Move Byte/Word	1 0 1 0 0 1 0 w			
CMPS = Compare Byte/Word	1 0 1 0 0 1 1 w			
SCAS = Scan Byte/Word	1 0 1 0 1 1 1 w			
LODS = Load Byte/Wd to AL/AX	1 0 1 0 1 1 0 w			
STOS = Stor Byte/Wd from AL/A	1 0 1 0 1 0 1 w			
CONTROL TRANSFER				
CALL = Call:				
Direct Within Segment	1 1 1 0 1 0 0 0	disp-low	disp-high	
Indirect Within Segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m		
Direct Intersegment	1 0 0 1 1 0 1 0	offset-low	offset-high	
		seg-low	seg-high	
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m		

80C86A/80C88A INSTRUCTION SET SUMMARY (Continued)

Mnemonic and Description	Instruction Code		
JMP = Unconditional Jump:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Direct Within Segment	1 1 1 0 1 0 0 1	disp-low	disp-high
Direct Within Segment-Short	1 1 1 0 1 0 1 1	disp	
Indirect Within Segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	
Direct Intersegment	1 1 1 0 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	
RET = Return from CALL:			
Within Segment	1 1 0 0 0 0 1 1		
Within Seg Adding Immed to SP	1 1 0 0 0 0 1 0	data-low	data-high
Intersegment	1 1 0 0 1 0 1 1		
Intersegment Adding Immediate to SP	1 1 0 0 1 0 1 0	data-low	data-high
JE/JZ = Jump on Equal/Zero	0 1 1 1 0 1 0 0	disp	
JL/JNGE = Jump on Less/Not Greater or Equal	0 1 1 1 1 1 0 0	disp	
JLE/JNG = Jump on Less or Equal/Not Greater	0 1 1 1 1 1 1 0	disp	
JB/JNAE = Jump on Below/Not Above or Equal	0 1 1 1 0 0 1 0	disp	
JBE/JNA = Jump on Below or Equal/Not Above	0 1 1 1 0 1 1 0	disp	
JP/JPE = Jump on Parity/Parity Even	0 1 1 1 1 0 1 0	disp	
JO = Jump on Overflow	0 1 1 1 0 0 0 0	disp	
JS = Jump on Sign	0 1 1 1 1 0 0 0	disp	
JNE/JNZ = Jump on Not Equal/Not Zero	0 1 1 1 0 1 0 1	disp	
JNL/JGE = Jump on Not Less/Greater or Equal	0 1 1 1 1 1 0 1	disp	
JNLE/JG = Jump on Not Less or Equal/Greater	0 1 1 1 1 1 1 1	disp	
JNB/JAE = Jump on Not Below/Above or Equal	0 1 1 1 0 0 1 1	disp	
JNBE/JA = Jump on Not Below or Equal/Above	0 1 1 1 0 1 1 1	disp	
JNP/JPO = Jump on Not Par/Par Odd	0 1 1 1 1 0 1 1	disp	
JNO = Jump on Not Overflow	0 1 1 1 0 0 0 1	disp	
JNS = Jump on Not Sign	0 1 1 1 1 0 0 1	disp	
LOOP = Loop CX Times	1 1 1 0 0 0 1 0	disp	
LOOPZ/LOOPE = Loop While Zero/Equal	1 1 1 0 0 0 0 1	disp	
LOOPNZ/LOOPNE = Loop While Not Zero/Equal	1 1 1 0 0 0 0 0	disp	
JCZX = Jump on CX Zero	1 1 1 0 0 0 1 1	disp	
INT = Interrupt			
Type Specified	1 1 0 0 1 1 0 1	type	
Type 3	1 1 0 0 1 1 0 0		
INTO = Interrupt on Overflow	1 1 0 0 1 1 1 0		
IRET = Interrupt Return	1 1 0 0 1 1 1 1		

80C86A/80C88A INSTRUCTION SET SUMMARY (Continued)

Mnemonic and Description	Instruction Code	
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
PROCESSOR CONTROL		
CLC = Clear Carry	1 1 1 1 1 0 0 0	
CMC = Complement Carry	1 1 1 1 0 1 0 1	
STC = Set Carry	1 1 1 1 1 0 0 1	
CLD = Clear Direction	1 1 1 1 1 1 0 0	
STD = Set Direction	1 1 1 1 1 1 0 1	
CLI = Clear Interrupt	1 1 1 1 1 0 1 0	
STI = Set Interrupt	1 1 1 1 1 0 1 1	
HLT = Halt	1 1 1 1 0 1 0 0	
WAIT = Wait	1 0 0 1 1 0 1 1	
ESC = Escape (to External Device)	1 1 0 1 1 x x x	mod x x x r/m
LOCK = Bus Lock Prefix	1 1 1 1 0 0 0 0	

NOTES:

AL = 8-bit accumulator
 AX = 16-bit accumulator
 CX = Count register
 DS = Data segment
 ES = Extra segment
 Above/below refers to unsigned value.
 Greater = more positive;
 Less = less positive (more negative) signed values
 if d = 1 then "to" reg; if d = 0 then "from" reg
 if w = 1 then word instruction; if w = 0 then byte instruction
 if mod = 11 then r/m is treated as a REG field
 if mod = 00 then DISP = 0*, disp-low and disp-high are absent
 if mod = 01 then DISP = disp-low sign-extended to 16 bits, disp-high is absent
 if mod = 10 then DISP = disp-high: disp-low
 if r/m = 000 then EA = (BX) + (SI) + DISP
 if r/m = 001 then EA = (BX) + (DI) + DISP
 if r/m = 010 then EA = (BP) + (SI) + DISP
 if r/m = 011 then EA = (BP) + (DI) + DISP
 if r/m = 100 then EA = (SI) + DISP
 if r/m = 101 then EA = (DI) + DISP
 if r/m = 110 then EA = (BP) + DISP*
 if r/m = 111 then EA = (BX) + DISP
 DISP follows 2nd byte of instruction (before data if required)
 *except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.
 **MOV CS, REG/MEMORY not allowed.

if s:w = 01 then 16 bits of immediate data form the operand.
 if s:w = 11 then an immediate data byte is sign extended to form the 16-bit operand.
 if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
 x = don't care
 z is used for string primitives for comparison with ZF FLAG.

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:

FLAGS =
 X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

Mnemonics © Intel, 1978

DATA SHEET REVISION REVIEW

The following list represents key differences between this and the -001 data sheet. Please review this summary carefully.

1. In the Pin Description Table (Table 1), the description of the HLDA signal being issued has been corrected. HLDA will be issued in the middle of either the T4 or Ti state.



8087 MATH COPROCESSOR

- Adds Arithmetic, Trigonometric, Exponential, and Logarithmic Instructions to the Standard 8086/8088 and 80186/80188 Instruction Set for All Data Types
 - CPU/8087 Supports 7 Data Types: 16-, 32-, 64-Bit Integers, 32-, 64-, 80-Bit Floating Point, and 18-Digit BCD Operands
 - Compatible with IEEE Floating Point Standard 754
- Available in 5 MHz (8087), 8 MHz (8087-2) and 10 MHz (8087-1): 8 MHz 80186/80188 System Operation Supported with the 8087-1
 - Adds 8 x 80-Bit Individually Addressable Register Stack to the 8086/8088 and 80186/80188 Architecture
 - 7 Built-In Exception Handling Functions
 - MULTIBUS® System Compatible Interface

The Intel 8087 Math CoProcessor is an extension to the Intel 8086/8088 microprocessor architecture. When combined with the 8086/8088 microprocessor, the 8087 dramatically increases the processing speed of computer applications which utilize mathematical operations such as CAM, numeric controllers, CAD or graphics.

The 8087 Math CoProcessor adds 68 mnemonics to the 8086 microprocessor instruction set. Specific 8087 math operations include logarithmic, arithmetic, exponential, and trigonometric functions. The 8087 supports integer, floating point and BCD data formats, and fully conforms to the ANSI/IEEE floating point standard.

The 8087 is fabricated with HMOS III technology and packaged in a 40-pin cerdip package.

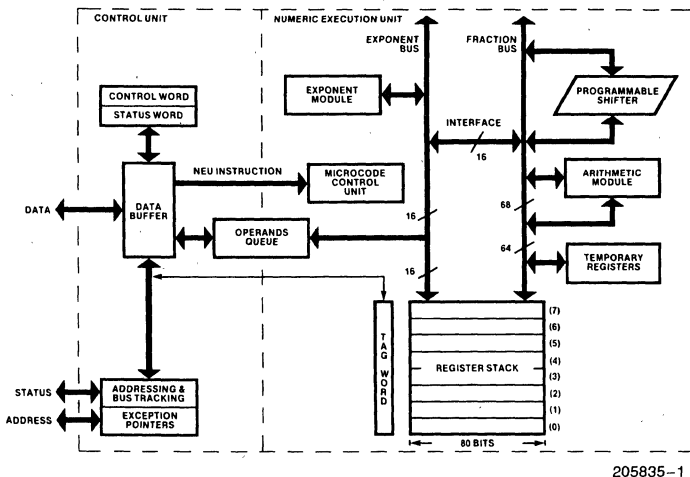


Figure 1. 8087 Block Diagram

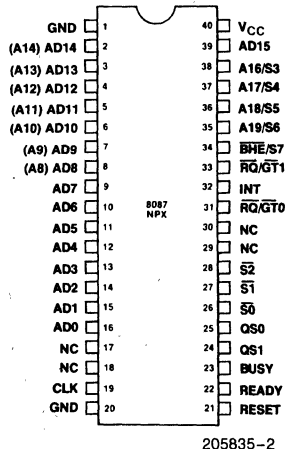


Figure 2. 8087 Pin Configuration

Table 1. 8087 Pin Description

Symbol	Type	Name and Function																														
AD15–AD0	I/O	ADDRESS DATA: These lines constitute the time multiplexed memory address (T_1) and data (T_2 , T_3 , T_W , T_4) bus. A0 is analogous to the \overline{BHE} for the lower byte of the data bus, pins D7–D0. It is LOW during T_1 when a byte is to be transferred on the lower portion of the bus in memory operations. Eight-bit oriented devices tied to the lower half of the bus would normally use A0 to condition chip select functions. These lines are active HIGH. They are input/output lines for 8087-driven bus cycles and are inputs which the 8087 monitors when the CPU is in control of the bus. A15–A8 do not require an address latch in an 8088/8087 or 80188/8087. The 8087 will supply an address for the T_1 – T_4 period.																														
A19/S6, A18/S5, A17/S4, A16/S3	I/O	ADDRESS MEMORY: During T_1 these are the four most significant address lines for memory operations. During memory operations, status information is available on these lines during T_2 , T_3 , T_W , and T_4 . For 8087-controlled bus cycles, S6, S4, and S3 are reserved and currently one (HIGH), while S5 is always LOW. These lines are inputs which the 8087 monitors when the CPU is in control of the bus.																														
$\overline{BHE}/S7$	I/O	BUS HIGH ENABLE: During T_1 the bus high enable signed (\overline{BHE}) should be used to enable data onto the most significant half of the data bus, pins D15–D8. Eight-bit-oriented devices tied to the upper half of the bus would normally use \overline{BHE} to condition chip select functions. \overline{BHE} is LOW during T_1 for read and write cycles when a byte is to be transferred on the high portion of the bus. The S7 status information is available during T_2 , T_3 , T_W , and T_4 . The signal is active LOW. S7 is an input which the 8087 monitors during the CPU-controlled bus cycles.																														
$\overline{S2}$, $\overline{S1}$, $\overline{S0}$	I/O	<p>STATUS: For 8087-driven, these status lines are encoded as follows:</p> <table border="1"> <thead> <tr> <th></th> <th>$\overline{S2}$</th> <th>$\overline{S1}$</th> <th>$\overline{S0}$</th> <th></th> </tr> </thead> <tbody> <tr> <td>0 (LOW)</td> <td>X</td> <td>X</td> <td></td> <td>Unused</td> </tr> <tr> <td>1 (HIGH)</td> <td>0</td> <td>0</td> <td></td> <td>Unused</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td></td> <td>Read Memory</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td></td> <td>Write Memory</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td></td> <td>Passive</td> </tr> </tbody> </table> <p>Status is driven active during T_4, remains valid during T_1 and T_2, and is returned to the passive state (1, 1, 1) during T_3 or during T_W when READY is HIGH. This status is used by the 8288 Bus Controller (or the 82188 Integrated Bus Controller with an 80186/80188 CPU) to generate all memory access control signals. Any change in $\overline{S2}$, $\overline{S1}$, or $\overline{S0}$ during T_4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle. These signals are monitored by the 8087 when the CPU is in control of the bus.</p>		$\overline{S2}$	$\overline{S1}$	$\overline{S0}$		0 (LOW)	X	X		Unused	1 (HIGH)	0	0		Unused	1	0	1		Read Memory	1	1	0		Write Memory	1	1	1		Passive
	$\overline{S2}$	$\overline{S1}$	$\overline{S0}$																													
0 (LOW)	X	X		Unused																												
1 (HIGH)	0	0		Unused																												
1	0	1		Read Memory																												
1	1	0		Write Memory																												
1	1	1		Passive																												
$\overline{RQ}/\overline{GT0}$	I/O	<p>REQUEST/GRANT: This request/grant pin is used by the 8087 to gain control of the local bus from the CPU for operand transfers or on behalf of another bus master. It must be connected to one of the two processor request/grant pins. The request/grant sequence on this pin is as follows:</p> <ol style="list-style-type: none"> 1. A pulse one clock wide is passed to the CPU to indicate a local bus request by either the 8087 or the master connected to the 8087 $\overline{RQ}/\overline{GT1}$ pin. 2. The 8087 waits for the grant pulse and when it is received will either initiate bus transfer activity in the clock cycle following the grant or pass the grant out on the $\overline{RQ}/\overline{GT1}$ pin in this clock if the initial request was for another bus master. 3. The 8087 will generate a release pulse to the CPU one clock cycle after the completion of the last 8087 bus cycle or on receipt of the release pulse from the bus master on $\overline{RQ}/\overline{GT1}$. <p>For 80186/80188 systems the same sequence applies except $\overline{RQ}/\overline{GT}$ signals are converted to appropriate HOLD, HLDA signals by the 82188 Integrated Bus Controller. This is to conform with 80186/80188's HOLD, HLDA bus exchange protocol. Refer to the 82188 data sheet for further information.</p>																														

Table 1. 8087 Pin Description (Continued)

Symbol	Type	Name and Function															
$\overline{RQ}/\overline{GT}1$	I/O	<p>REQUEST/GRANT: This request/grant pin is used by another local bus master to force the 8087 to request the local bus. If the 8087 is not in control of the bus when the request is made the request/grant sequence is passed through the 8087 on the $\overline{RQ}/\overline{GT}0$ pin one cycle later. Subsequent grant and release pulses are also passed through the 8087 with a two and one clock delay, respectively, for resynchronization. $\overline{RQ}/\overline{GT}1$ has an internal pullup resistor, and so may be left unconnected. If the 8087 has control of the bus the request/grant sequence is as follows:</p> <ol style="list-style-type: none"> 1. A pulse 1 CLK wide from another local bus master indicates a local bus request to the 8087 (pulse 1). 2. During the 8087's next T_4 or T_1 a pulse 1 CLK wide from the 8087 to the requesting master (pulse 2) indicates that the 8087 has allowed the local bus to float and that it will enter the "RQ/GT acknowledge" state at the next CLK. The 8087's control unit is disconnected logically from the local bus during "RQ/GT acknowledge." 3. A pulse 1 CLK wide from the requesting master indicates to the 8087 (pulse 3) that the "RQ/GT" request is about to end and that the 8087 can reclaim the local bus at the next CLK. <p>Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW. For 80186/80188 system, the $\overline{RQ}/\overline{GT}1$ line may be connected to the 82188 Integrated Bus Controller. In this case, a third processor with a HOLD, HLDA bus exchange system may acquire the bus from the 8087. For this configuration, $\overline{RQ}/\overline{GT}1$ will only be used if the 8087 is the bus master. Refer to 82188 data sheet for further information.</p>															
QS1, QS0	I	<p>QS1, QS0: QS1 and QS0 provide the 8087 with status to allow tracking of the CPU instruction queue.</p> <table border="0"> <tr> <td style="text-align: center;">QS1</td> <td style="text-align: center;">QS0</td> <td></td> </tr> <tr> <td>0 (LOW)</td> <td>0</td> <td>No Operation</td> </tr> <tr> <td>0</td> <td>1</td> <td>First Byte of Op Code from Queue</td> </tr> <tr> <td>1 (HIGH)</td> <td>0</td> <td>Empty the Queue</td> </tr> <tr> <td>1</td> <td>1</td> <td>Subsequent Byte from Queue</td> </tr> </table>	QS1	QS0		0 (LOW)	0	No Operation	0	1	First Byte of Op Code from Queue	1 (HIGH)	0	Empty the Queue	1	1	Subsequent Byte from Queue
QS1	QS0																
0 (LOW)	0	No Operation															
0	1	First Byte of Op Code from Queue															
1 (HIGH)	0	Empty the Queue															
1	1	Subsequent Byte from Queue															
INT	O	<p>INTERRUPT: This line is used to indicate that an unmasked exception has occurred during numeric instruction execution when 8087 interrupts are enabled. This signal is typically routed to an 8259A for 8086/8088 systems and to INTO for 80186/80188 systems. INT is active HIGH.</p>															
BUSY	O	<p>BUSY: This signal indicates that the 8087 NEU is executing a numeric instruction. It is connected to the CPU's TEST pin to provide synchronization. In the case of an unmasked exception BUSY remains active until the exception is cleared. BUSY is active HIGH.</p>															
READY	I	<p>READY: READY is the acknowledgement from the addressed memory device that it will complete the data transfer. The RDY signal from memory is synchronized by the 8284A Clock Generator to form READY for 8086 systems. For 80186/80188 systems, RDY is synchronized by the 82188 Integrated Bus Controller to form READY. This signal is active HIGH.</p>															
RESET	I	<p>RESET: RESET causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. RESET is internally synchronized.</p>															
CLK	I	<p>CLOCK: The clock provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.</p>															
V_{CC}		<p>POWER: V_{CC} is the +5V power supply pin.</p>															
GND		<p>GROUND: GND are the ground pins.</p>															

NOTE:

For the pin descriptions of the 8086, 8088, 80186 and 80188 CPUs, reference the respective data sheets (8086, 8088, 80186, 80188).

APPLICATION AREAS

The 8087 provides functions meant specifically for high performance numeric processing requirements. Trigonometric, logarithmic, and exponential functions are built into the coprocessor hardware. These functions are essential in scientific, engineering, navigational, or military applications.

The 8087 also has capabilities meant for business or commercial computing. An 8087 can process Binary Coded Decimal (BCD) numbers up to 18 digits without roundoff errors. It can also perform arithmetic on integers as large as 64 bits $\pm 10^{18}$.

PROGRAMMING LANGUAGE SUPPORT

Programs for the 8087 can be written in Intel's high-level languages for 8086/8088 and 80186/80188 Systems; ASM-86 (the 8086, 8088 assembly language), PL/M-86, FORTRAN-86, and PASCAL-86.

RELATED INFORMATION

For 8086, 8088, 80186 or 80188 details, refer to the respective data sheets. For 80186 or 80188 systems, also refer to the 82188 Integrated Bus Controller data sheet.

FUNCTIONAL DESCRIPTION

The 8087 Math CoProcessor's architecture is designed for high performance numeric computing in conjunction with general purpose processing.

The 8087 is a numeric processor extension that provides arithmetic and logical instruction support for a variety of numeric data types. It also executes numerous built-in transcendental functions (e.g., tangent and log functions). The 8087 executes instructions as a coprocessor to a maximum mode CPU. It effectively extends the register and instruction set of the system and adds several new data types as well. Figure 3 presents the registers of the CPU + 8087. Table 2 shows the range of data types supported by the 8087. The 8087 is treated as an extension to the CPU, providing register, data types, control, and instruction capabilities at the hardware level. At the programmer's level the CPU and the 8087 are viewed as a single unified processor.

System Configuration

As a coprocessor to an 8086 or 8088, the 8087 is wired in parallel with the CPU as shown in Figure 4. Figure 5 shows the 80186/80188 system configuration. The CPU's status (S0-S2) and queue status lines (QS0-QS1) enable the 8087 to monitor and decode instructions in synchronization with the CPU and without any CPU overhead. For 80186/80188 systems, the queue status signals of the 80186/80188 are synchronized to 8087 requirements by the 8288 Integrated Bus Controller. Once started, the 8087 can process in parallel with, and independent of, the host CPU. For resynchronization, the 8087's BUSY signal informs the CPU that the 8087 is executing an instruction and the CPU WAIT instruction tests this signal to insure that the 8087 is ready to execute subsequent instructions. The 8087 can interrupt the CPU when it detects an error or exception. The 8087's interrupt request line is typically routed to the CPU through an 8259A Programmable Interrupt Controller for 8086, 8088 systems and INT0 for 80186/80188.

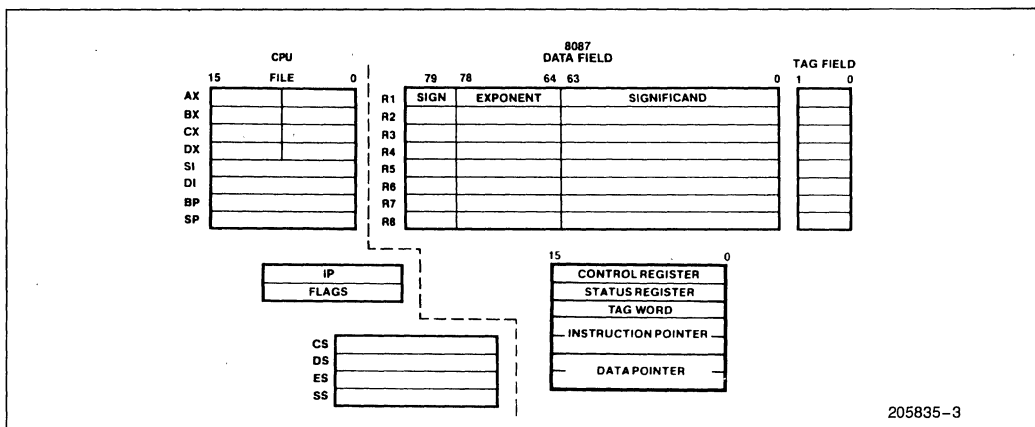


Figure 3. CPU + 8087 Architecture

The 8087 uses one of the request/grant lines of the 8086/8088 architecture (typically $\overline{RQ}/\overline{GT0}$) to obtain control of the local bus for data transfers. The other request/grant line is available for general system use (for instance by an I/O processor in LOCAL mode). A bus master can also be connected to the 8087's $\overline{RQ}/\overline{GT1}$ line. In this configuration the 8087 will pass the request/grant handshake signals between the CPU and the attached master when the 8087 is not in control of the bus and will relinquish the bus to the master directly when the 8087 is in control. In this way two additional masters can be configured in an 8086/8088 system; one will share the 8086/8088 bus with the 8087 on a first-come first-served basis, and the second will be guaranteed to be higher in priority than the 8087.

For 80186/80188 systems, $\overline{RQ}/\overline{GT0}$ and $\overline{RQ}/\overline{GT1}$ are connected to the corresponding inputs of the 82188 Integrated Bus Controller. Because the 80186/80188 has a HOLD, HLDA bus exchange protocol, an interface is needed which will translate $\overline{RQ}/\overline{GT}$ signals to corresponding HOLD, HLDA signals and vice versa. One of the functions of the 82188 IBC is to provide this translation. $\overline{RQ}/\overline{GT0}$ is translated to HOLD, HLDA signals which are then directly connected to the 80186/80188. The $\overline{RQ}/\overline{GT1}$ line is also translated into HOLD, HLDA signals (referred to as SYSHOLD, SYSHLDA signals) by the 82188 IBC. This allows a third processor (using a HOLD, HLDA bus exchange protocol) to gain control of the bus.

Unlike an 8086/8087 system, $\overline{RQ}/\overline{GT}$ is only used when the 8087 has bus control. If the third processor requests the bus when the current bus master is the 80186/80188, the 82188 IBC will directly pass the request onto the 80186/80188 without going through the 8087. The third processor has the highest bus priority in the system. If the 8087 requests the bus while the third processor has bus control, the grant pulse will not be issued until the third processor releases the bus (using SYSHOLD). In this configuration, the third processor has the highest priority, the 8087 has the next highest, and the 80186/80188 has the lowest bus priority.

Bus Operation

The 8087 bus structure, operation and timing are identical to all other processors in the 8086/8088 series (maximum mode configuration). The address is time multiplexed with the data on the first 16/8 lines of the address/data bus. A16 through A19 are time multiplexed with four status lines S3-S6. S3, S4 and S6 are always one (HIGH) for 8087-driven bus cycles while S5 is always zero (LOW). When the 8087 is monitoring CPU bus cycles (passive mode) S6 is also monitored by the 8087 to differentiate 8086/8088 activity from that of a local I/O processor or any other local bus master. (The 8086/8088 must be the only processor on the local bus to drive S6 LOW). S7 is multiplexed with and has the same value as BHE for all 8087 bus cycles.

Table 2. 8087 Data Types

Data Formats	Range	Precision	Most Significant Byte									
			7	07	07	07	07	07	07	07	07	0
Word Integer	10 ⁴	16 Bits	I ₁₅ I ₀ Two's Complement									
Short Integer	10 ⁹	32 Bits	I ₃₁ I ₀ Two's Complement									
Long Integer	10 ¹⁸	64 Bits	I ₆₃ I ₀ Two's Complement									
Packed BCD	10 ¹⁸	18 Digits	S ₁ D ₁₇ D ₁₆ D ₁ D ₀									
Short Real	10 ^{±38}	24 Bits	S ₁ E ₇ E ₀ F ₁ F ₂₃ F ₀ Implicit									
Long Real	10 ^{±308}	53 Bits	S ₁ E ₁₀ E ₀ F ₁ F ₅₂ F ₀ Implicit									
Temporary Real	10 ^{±4932}	64 Bits	S ₁ E ₁₄ E ₀ F ₀ F ₆₃									

Integer: I

Packed BCD: (-1)^S(D₁₇...D₀)

Real: (-1)^S(2^{E-Bias})(F₀*F₁...)

bias = 127 for Short Real

1023 for Long Real

16383 for Temp Real

The first three status lines, $\overline{S0}$ – $\overline{S2}$, are used with an 8288 bus controller or 82188 Integrated Bus Controller to determine the type of bus cycle being run:

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	
0	X	X	Unused
1	0	0	Unused
1	0	1	Memory Data Read
1	1	0	Memory Data Write
1	1	1	Passive (no bus cycle)

Programming Interface

The 8087 includes the standard 8086, 8088 instruction set for general data manipulation and program control. It also includes 68 numeric instructions for extended precision integer, floating point, trigonometric, logarithmic, and exponential functions. Sample execution times for several 8087 functions are shown in Table 3. Overall performance is up to 100 times that of an 8086 processor for numeric instructions.

Any instruction executed by the 8087 is the combined result of the CPU and 8087 activity. The CPU and the 8087 have specialized functions and registers providing fast concurrent operation. The CPU controls overall program execution while the 8087 uses the coprocessor interface to recognize and perform numeric operations.

Table 2 lists the seven data types the 8087 supports and presents the format for each type. Internally, the 8087 holds all numbers in the temporary real format. Load and store instructions automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating point numbers or 18-digit packed BCD numbers into temporary real format and vice versa. The 8087 also provides the capability to control round off, underflow, and overflow errors in each calculation.

Computations in the 8087 use the processor's register stack. These eight 80-bit registers provide the equivalent capacity of 20 32-bit registers. The 8087 register set can be accessed as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers.

Table 5 lists the 8087's instructions by class. All appear as ESCAPE instructions to the host. Assembly language programs are written in ASM-86, the 8086, 8088 assembly language.

Table 3. Execution Times for Selected 8086/8087 Numeric Instructions and Corresponding 8086 Emulation

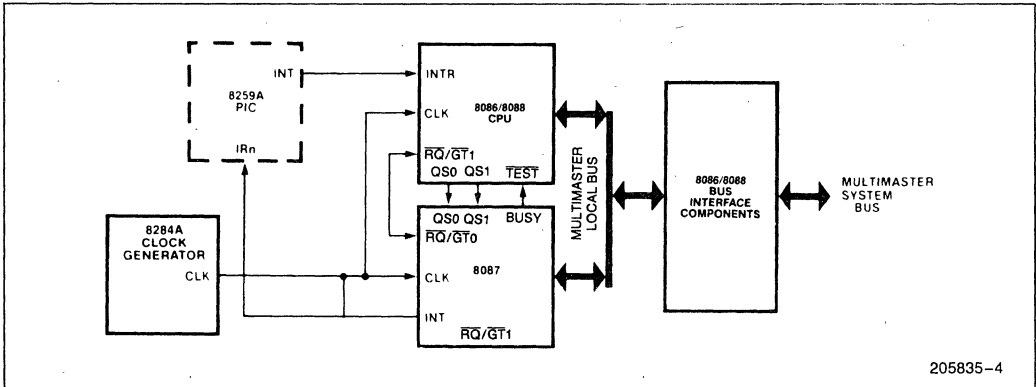
Floating Point Instruction	Approximate Execution Time (μ s)	
	8086/8087 (8 MHz Clock)	8086 Emulation
Add/Subtract	10.6	1000
Multiply (Single Precision)	11.9	1000
Multiply (Extended Precision)	16.9	1312
Divide	24.4	2000
Compare	-5.6	812
Load (Double Precision)	-6.3	1062
Store (Double Precision)	13.1	750
Square Root	22.5	12250
Tangent	56.3	8125
Exponentiation	62.5	10687

NUMERIC PROCESSOR EXTENSION ARCHITECTURE

As shown in Figure 1, the 8087 is internally divided into two processing elements, the control unit (CU) and the numeric execution unit (NEU). The NEU executes all numeric instructions, while the CU receives and decodes instructions, reads and writes memory operands and executes 8087 control instructions. The two elements are able to operate independently of one another, allowing the CU to maintain synchronization with the CPU while the NEU is busy processing a numeric instruction.

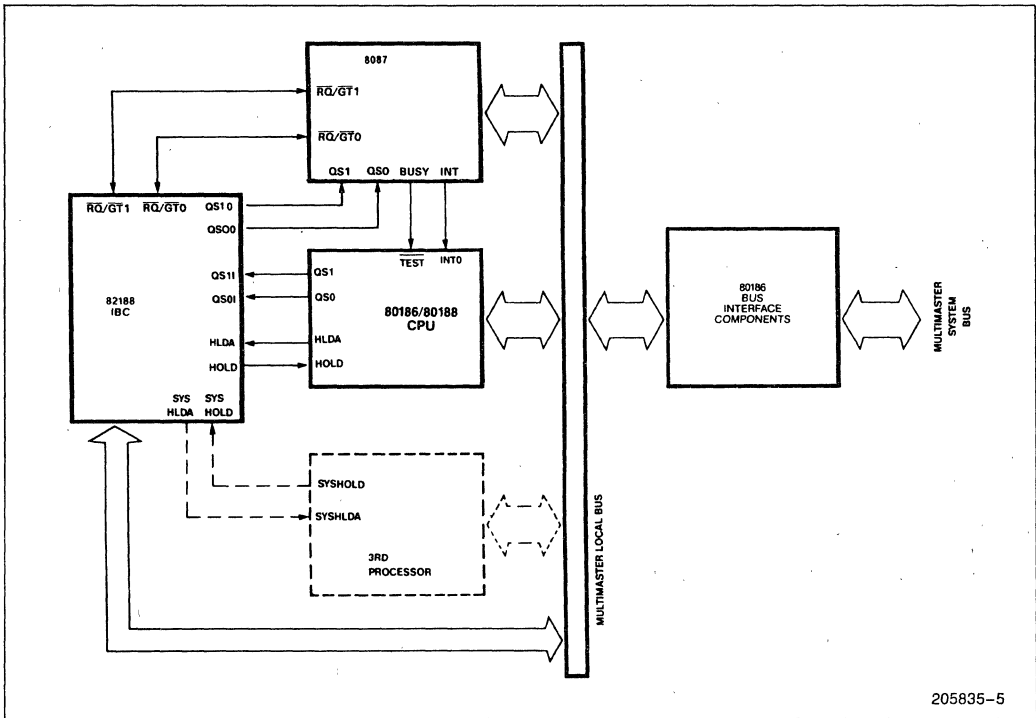
Control Unit

The CU keeps the 8087 operating in synchronization with its host CPU. 8087 instructions are intermixed with CPU instructions in a single instruction stream. The CPU fetches all instructions from memory; by monitoring the status ($\overline{S0}$ – $\overline{S2}$, $\overline{S6}$) emitted by the CPU, the control unit determines when an instruction is being fetched. The CPU monitors the data bus in parallel with the CPU to obtain instructions that pertain to the 8087.



205835-4

Figure 4. 8086/8087, 8088/8087 System Configuration



205835-5

Figure 5. 80186/8087, 80188/8087 System Configuration

The CU maintains an instruction queue that is identical to the queue in the host CPU. The CU automatically determines if the CPU is an 8086/80186 or an 8088/80188 immediately after reset (by monitoring the $\overline{BHE}/S7$ line) and matches its queue length accordingly. By monitoring the CPU's queue status lines (QS0, QS1), the CU obtains and decodes instructions from the queue in synchronization with the CPU.

A numeric instruction appears as an ESCAPE instruction to the CPU. Both the CPU and 8087 decode and execute the ESCAPE instruction together. The 8087 only recognizes the numeric instructions shown in Table 5. The start of a numeric operation is accomplished when the CPU executes the ESCAPE instruction. The instruction may or may not identify a memory operand.

The CPU does, however, distinguish between ESC instructions that reference memory and those that do not. If the instruction refers to a memory operand, the CPU calculates the operand's address using any one of its available addressing modes, and then performs a "dummy read" of the word at that location. (Any location within the 1M byte address space is allowed.) This is a normal read cycle except that the CPU ignores the data it receives. If the ESC instruction does not contain a memory reference (e.g. an 8087 stack operation), the CPU simply proceeds to the next instruction.

An 8087 instruction can have one of three memory reference options: (1) not reference memory; (2) load an operand word from memory into the 8087; or (3) store an operand word from the 8087 into memory. If no memory reference is required, the 8087 simply executes its instruction. If a memory reference is required, the CU uses a "dummy read" cycle initiated by the CPU to capture and save the address that the CPU places on the bus. If the instruction is a load, the CU additionally captures the data word when it becomes available on the local data bus. If data required is longer than one word, the CU immediately obtains the bus from the CPU using the request/grant protocol and reads the rest of the information in consecutive bus cycles. In a store operation, the CU captures and saves the store address as in a load, and ignores the data word that follows in the "dummy read" cycle. When the 8087 is ready to perform the store, the CU obtains the bus from the CPU and writes the operand starting at the specified address.

Numeric Execution Unit

The NEU executes all instructions that involve the register stack; these include arithmetic, logical, transcendental, constant and data transfer instructions. The data path in the NEU is 84 bits wide (68 fractions bits, 15 exponent bits and a sign bit) which allows internal operand transfers to be performed at very high speeds.

When the NEU begins executing an instruction, it activates the 8087 BUSY signal. This signal can be used in conjunction with the CPU WAIT instruction to resynchronize both processors when the NEU has completed its current instruction.

Register Set

The CPU+8087 register set is shown in Figure 3. Each of the eight data registers in the 8087's register stack is 80 bits and is divided into "fields" corresponding to the 8087's temporary real data type.

At a given point in time the TOP field in the control word identifies the current top-of-stack register. A "push" operation decrements TOP by 1 and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by 1. Like CPU stacks in memory, the 8087 register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the top of the stack. These instructions implicitly address the register pointed to by the TOP. Other instructions allow the programmer to explicitly specify the register which is to be used. Explicit register addressing is "top-relative."

Status Word

The status word shown in Figure 6 reflects the overall state of the 8087; it may be stored in memory and then inspected by CPU code. The status word is a 16-bit register divided into fields as shown in Figure 6. The busy bit (bit 15) indicates whether the NEU is either executing an instruction or has an interrupt request pending (B=1), or is idle (B=0). Several instructions which store and manipulate the status word are executed exclusively by the CU, and these do not set the busy bit themselves.

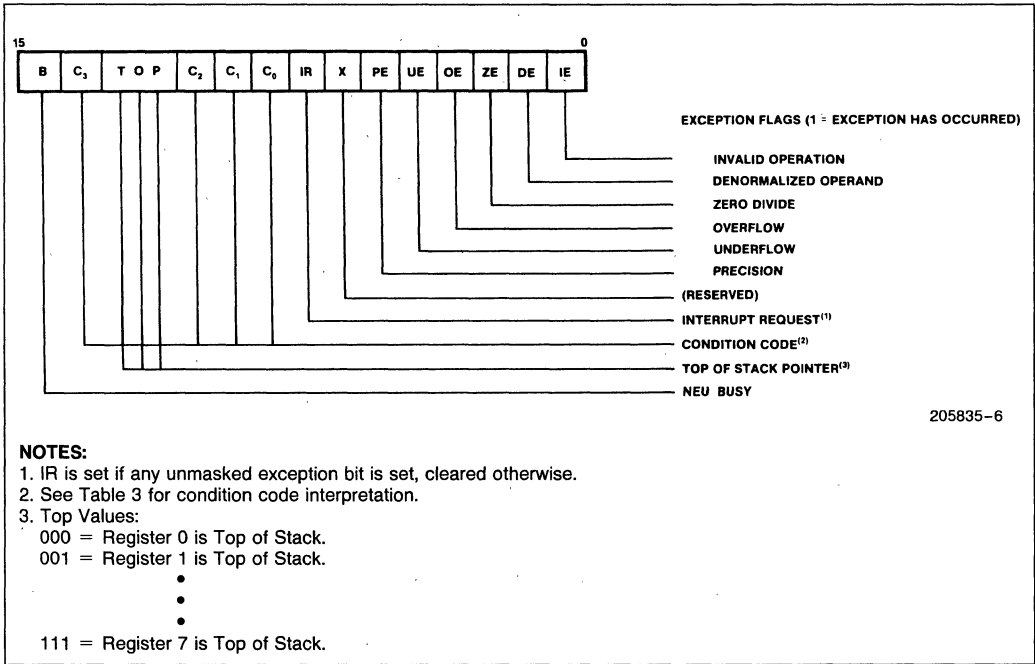


Figure 6. 8087 Status Word

The four numeric condition code bits (C₀–C₃) are similar to flags in a CPU: various instructions update these bits to reflect the outcome of the 8087 operations. The effect of these instructions on the condition code bits is summarized in Table 4.

Bits 14–12 of the status word point to the 8087 register that is the current top-of-stack (TOP) as described above.

Bit 7 is the interrupt request bit. This bit is set if any unmasked exception bit is set and cleared otherwise.

Bits 5–0 are set to indicate that the NEU has detected an exception while executing an instruction.

Tag Word

The tag word marks the content of each register as shown in Figure 7. The principal function of the tag word is to optimize the 8087's performance. The tag word can be used, however, to interpret the contents of 8087 registers.

Instruction and Data Pointers

The instruction and data pointers (see Figure 8) are provided for user-written error handlers. Whenever the 8087 executes a math instruction, the CU saves the instruction address, the operand address (if present) and the instruction opcode. 8087 instructions can store this data into memory.

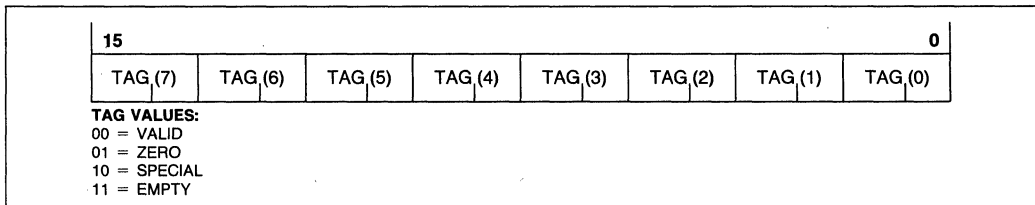


Figure 7. 8087 Tag Word

Table 4a. Condition Code Interpretation

Instruction Type	C ₃	C ₂	C ₁	C ₀	Interpretation
Compare, Test	0	0	X	0	ST > Source or 0 (FTST)
	0	0	X	1	ST < Source or 0 (FTST)
	1	0	X	0	ST = Source or 0 (FTST)
	1	1	X	1	ST is not comparable
Remainder	Q ₁	0	Q ₀	Q ₂	Complete reduction with three low bits of quotient (See Table 4b)
	U	1	U	U	Incomplete Reduction
Examine	0	0	0	0	Valid, positive unnormalized
	0	0	0	1	Invalid, positive, exponent = 0
	0	0	1	0	Valid, negative, unnormalized
	0	0	1	1	Invalid, negative, exponent = 0
	0	1	0	0	Valid, positive, normalized
	0	1	0	1	Infinity, positive
	0	1	1	0	Valid, negative, normalized
	0	1	1	1	Infinity, negative
	1	0	0	0	Zero, positive
	1	0	0	1	Empty
	1	0	1	0	Zero, negative
	1	0	1	1	Empty
	1	1	0	0	Invalid, positive, exponent = 0
	1	1	0	1	Empty
1	1	1	0	Invalid, negative, exponent = 0	
1	1	1	1	Empty	

NOTES:

1. ST = Top of stack
2. X = value is not affected by instruction
3. U = value is undefined following instruction
4. Q_n = Quotient bit n

Table 4b. Condition Code Interpretation after FPREM Instruction As a Function of Divided Value

Dividend Range	Q ₂	Q ₁	Q ₀
Dividend < 2 * Modulus	C ₃ ¹	C ₁ ¹	Q ₀
Dividend < 4 * Modulus	C ₃ ¹	Q ₁	Q ₀
Dividend ≥ 4 * Modulus	Q ₂	Q ₁	Q ₀

NOTE:

1. Previous value of indicated bit, not affected by FPREM instruction execution.

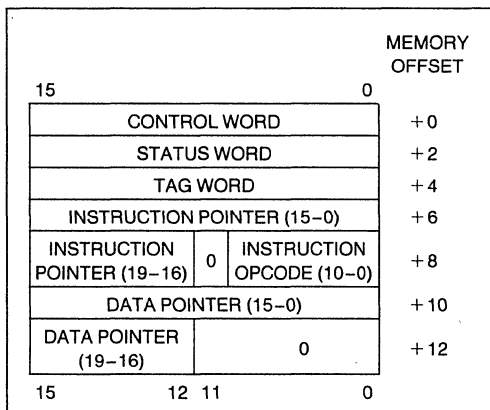


Figure 8. 8087 Instruction and Data Pointer Image in Memory

Control Word

The 8087 provides several processing options which are selected by loading a word from memory into the control word. Figure 9 shows the format and encoding of the fields in the control word.

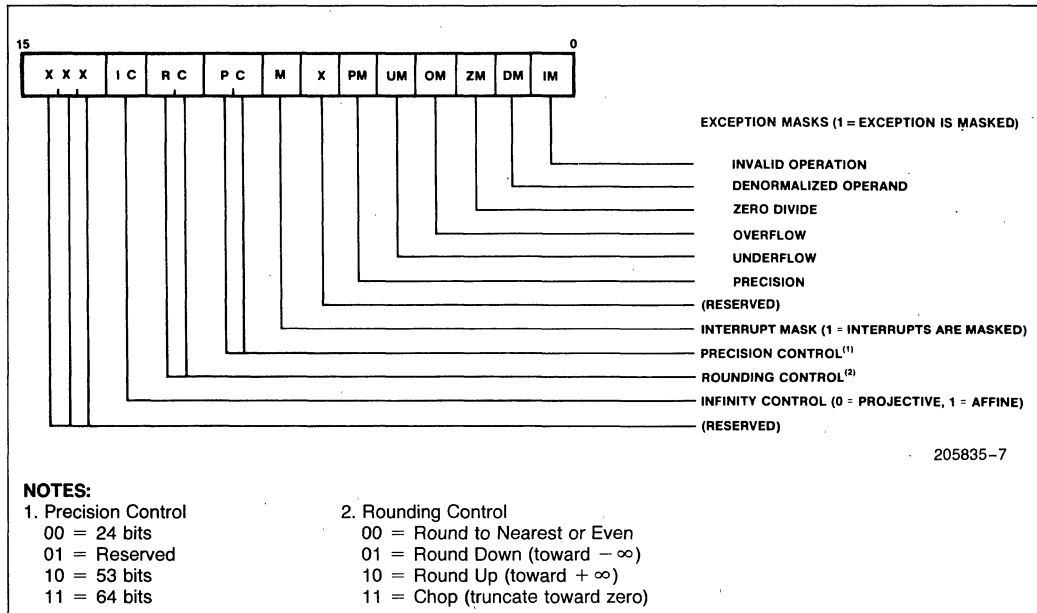
The low order byte of this control word configures 8087 interrupts and exception masking. Bits 5–0 of the control word contain individual masks for each of the six exceptions that the 8087 recognizes and bit 7 contains a general mask bit for all 8087 interrupts. The high order byte of the control word configures the 8087 operating mode including precision, rounding, and infinity controls. The precision control bits (bits 9–8) can be used to set the 8087 internal operating precision at less than the default of temporary real precision. This can be useful in providing compatibility with earlier generation arithmetic processors of smaller precision than the 8087. The rounding control bits (bits 11–10) provide for directed rounding and true chop as well as the unbiased round to nearest mode specified in the proposed IEEE standard. Control over closure of the number space at infinity is also provided (either affine closure, $\pm\infty$, or projective closure, ∞ , is treated as unsigned, may be specified).

Exception Handling

The 8087 detects six different exception conditions that can occur during instruction execution. Any or all exceptions will cause an interrupt if unmasked and interrupts are enabled.

If interrupts are disabled the 8087 will simply continue execution regardless of whether the host clears the exception. If a specific exception class is masked and that exception occurs, however, the 8087 will post the exception in the status register and perform an on-chip default exception handling procedure, thereby allowing processing to continue. The exceptions that the 8087 detects are the following:

1. INVALID OPERATION: Stack overflow, stack underflow, indeterminate form ($0/0$, $\infty - \infty$, etc.) or the use of a Non-Number (NaN) as an operand. An exponent value is reserved and any bit pattern with this value in the exponent field is termed a Non-Number and causes this exception. If this exception is masked, the 8087's default response is to generate a specific NaN called INDEFINITE, or to propagate already existing NaNs as the calculation result.



205835-7

Figure 9. 8087 Control Word

- 2. OVERFLOW: The result is too large in magnitude to fit the specified format. The 8087 will generate an encoding for infinity if this exception is masked.
- 3. ZERO DIVISOR: The divisor is zero while the dividend is a non-infinite, non-zero number. Again, the 8087 will generate an encoding for infinity if this exception is masked.
- 4. UNDERFLOW: The result is non-zero but too small in magnitude to fit in the specified format. If this exception is masked the 8087 will denormalize (shift right) the fraction until the exponent is in range. This process is called gradual underflow.

- 5. DENORMALIZED OPERAND: At least one of the operands or the result is denormalized; it has the smallest exponent but a non-zero significand. Normal processing continues if this exception is masked off.
- 6. INEXACT RESULT: If the true result is not exactly representable in the specified format, the result is rounded according to the rounding mode, and this flag is set. If this exception is masked, processing will simply continue.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias 0°C to 70°C
 Storage Temperature -65°C to +150°C
 Voltage on Any Pin with
 Respect to Ground -1.0V to +7V
 Power Dissipation 3.0 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $T_A = 0^\circ\text{C to } 70^\circ\text{C}$, $V_{CC} = 5V \pm 5\%$

Symbol	Parameter	Min	Max	Units	Test Conditions
V _{IL}	Input Low Voltage	-0.5	0.8	V	
V _{IH}	Input High Voltage	2.0	V _{CC} + 0.5	V	
V _{OL}	Output Low Voltage (Note 8)		0.45	V	I _{OL} = 2.5 mA
V _{OH}	Output High Voltage	2.4		V	I _{OH} = -400 μA
I _{CC}	Power Supply Current		475	mA	T _A = 25°C
I _{LI}	Input Leakage Current		± 10	μA	0V ≤ V _{IN} ≤ V _{CC}
I _{LO}	Output Leakage Current		± 10	μA	T _A = 25°C
V _{CL}	Clock Input Low Voltage	-0.5	0.6	V	
V _{CH}	Clock Input High Voltage	3.9	V _{CC} + 1.0	V	
C _{IN}	Capacitance of Inputs		10	pF	f _c = 1 MHz
C _{IO}	Capacitance of I/O Buffer (AD0-15, A16-A19, BHE, S2-S0, RQ/GT) and CLK		15	pF	f _c = 1 MHz
C _{OUT}	Capacitance of Outputs BUSY INT		10	pF	f _c = 1 MHz

A.C. CHARACTERISTICS $T_A = 0^\circ\text{C to } 70^\circ\text{C}$, $V_{CC} = 5\text{V} \pm 5\%$
TIMING REQUIREMENTS

Symbol	Parameter	8087		8087-2		8087-1 (See Note 7)		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
TCLCL	CLK Cycle Period	200	500	125	500	100	500	ns	
TCLCH	CLK Low Time	118		68		53		ns	
TCHCL	CLK High Time	69		44		39		ns	
TCH1CH2	CLK Rise Time		10		10		15	ns	From 1.0V to 3.5V
TCL2CL2	CLK Fall Time		10		10		15	ns	From 3.5V to 1.0V
TDVCL	Data In Setup Time	30		20		15		ns	
TCLDX	Data In Hold Time	10		10		10		ns	
TRYHCH	READY Setup Time	118		68		53		ns	
TCHRYX	READY Hold Time	30		20		5		ns	
TRYLCL	READY Inactive to CLK (Note 6)	-8		-8		-10		ns	
TGVCH	RQ/GT Setup Time (Note 8)	30		15		15		ns	
TCHGX	RQ/GT Hold Time	40		30		20		ns	
TQVCL	QS0-1 Setup Time (Note 8)	30		30		30		ns	
TCLQX	QS0-1 Hold Time	10		10		5		ns	
TSACH	Status Active Setup Time	30		30		30		ns	
TSNCL	Status Inactive Setup Time	30		30		30		ns	
TILIH	Input Rise Time (Except CLK)		20		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12		15	ns	From 2.0V to 0.8V

TIMING RESPONSES

Symbol	Parameter	8087		8087-2		8087-1 (See Note 7)		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
TCLML	Command Active Delay (Notes 1, 2)	10/0	35/70	10/0	35/70	10/0	35/70	ns	$C_L = 20-100\text{ pF}$ for all 8087 Outputs (in addition to 8087 self-load)
TCLMH	Command Inactive Delay (Notes 1, 2)	10/0	35/55	10/0	35/55	10/0	35/70	ns	
TRYHSH	Ready Active to Status Passive (Note 5)		110		65		45	ns	
TCHSV	Status Active Delay	10	110	10	60	10	45	ns	
TCLSH	Status Inactive Delay	10	130	10	70	10	55	ns	
TCLAV	Address Valid Delay	10	110	10	60	10	55	ns	
TCLAX	Address Hold Time	10		10		10		ns	

A.C. CHARACTERISTICS $T_A = 0^\circ\text{C to }70^\circ\text{C}$, $V_{CC} = 5V \pm 5\%$ (Continued)

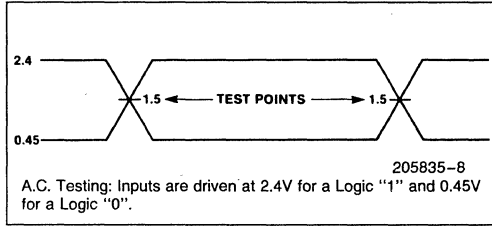
TIMING RESPONSES (Continued)

Symbol	Parameter	8087		8087-2		8087-1 (See Note 7)		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	TCLAX	45	ns	$C_L = 20\text{--}100\text{ pF}$ for all 8087 Outputs (in addition to 8087 self-load)
TSVLH	Status Valid to ALE High (Notes 1, 2)		15/30		15/30		15/30	ns	
TCLLH	CLK Low to ALE Valid (Notes 1, 2)		15/30		15/30		15/30	ns	
TCHLL	ALE Inactive Delay (Notes 1, 2)		15/30		15/30		15/30	ns	
TCLDV	Data Valid Delay	10	110	10	60	10	50	ns	
TCHDX	Status Hold Time	10		10		10	45	ns	
TCLDOX	Data Hold Time	10		10		10		ns	
TCVNV	Control Active Delay (Notes 1, 3)	5	45	5	45	5	45	ns	
TCVNX	Control Inactive Delay (Notes 1, 3)	10	45	10	45	10	45	ns	
TCHBV	BUSY and INT Valid Delay	10	150	10	85	10	65	ns	
TCHDTL	Direction Control Active Delay (Notes 1, 3)		50		50		50	ns	
TCHDTH	Direction Control Inactive Delay (Notes 1, 3)		30		30		30	ns	
TSVDTV	STATUS to DT/ \bar{R} Delay (Notes 1, 4)	0	30	0	30	0	30	ns	
TCLDTV	DT/ \bar{R} Active Delay (Notes 1, 4)	0	55	0	55	0	55	ns	
TCHDNV	\overline{DEN} Active Delay (Notes 1, 4)	0	55	0	55	0	55	ns	
TCHDNX	\overline{DEN} Inactive Delay (Notes 1, 4)	5	55	5	55	5	55	ns	
TCLGL	RQ/GT Active Delay (Note 8)	0	85	0	50	0	38	ns	$C_L = 40\text{ pF}$ (in addition to 8087 self-load)
TCLGH	RQ/GT Inactive Delay	0	85	0	50	0	45	ns	
TOLOH	Output Rise Time		20		20		15	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12		12	ns	From 2.0V to 0.8V

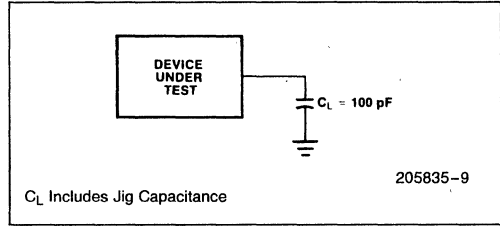
NOTES:

1. Signal at 8284A, 8288, or 82188 shown for reference only.
2. 8288 timing/82188 timing.
3. 8288 timing.
4. 82188 timing.
5. Applies only to T_3 and wait states.
6. Applies only to T_2 state (8 ns into T_3).
7. **IMPORTANT SYSTEM CONSIDERATION:** Some 8087-1 timing parameters are constrained relative to the corresponding 8086-1 specifications. Therefore, 8086-1 systems incorporating the 8087-1 should be designed with the 8087-1 specifications.
8. Changes since last revision.

A.C. TESTING INPUT, OUTPUT WAVEFORM

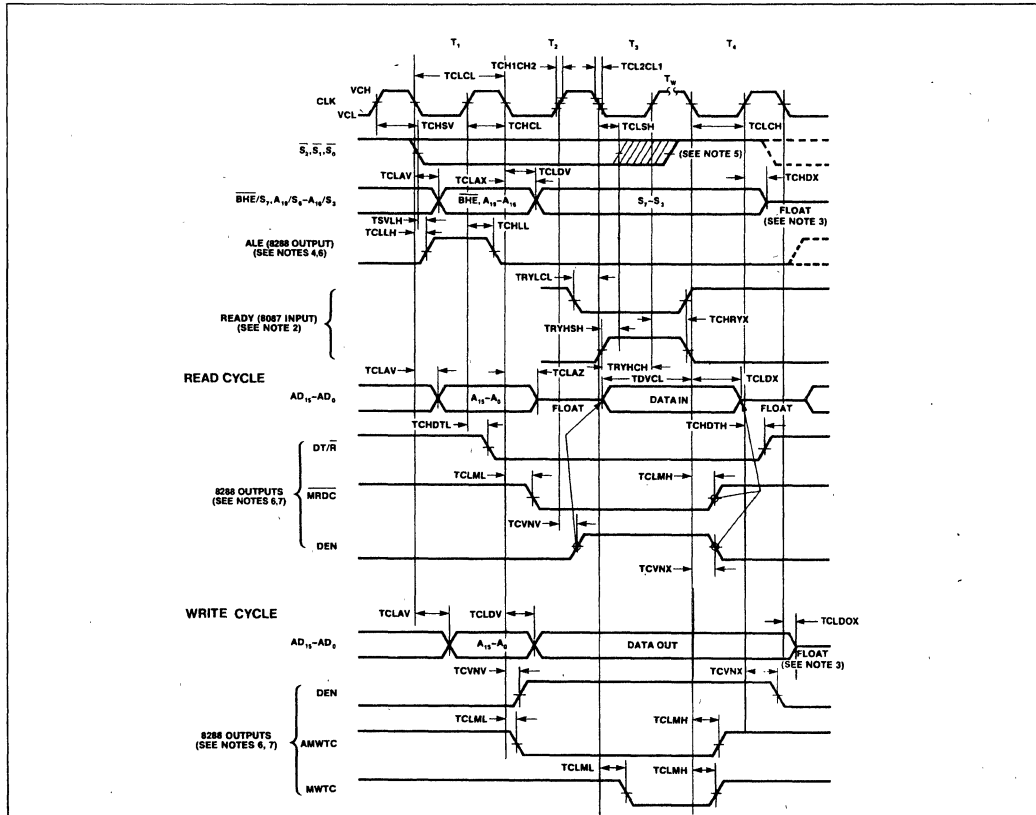


A.C. TESTING LOAD CIRCUIT



WAVEFORMS

MASTER MODE (with 8288 references)



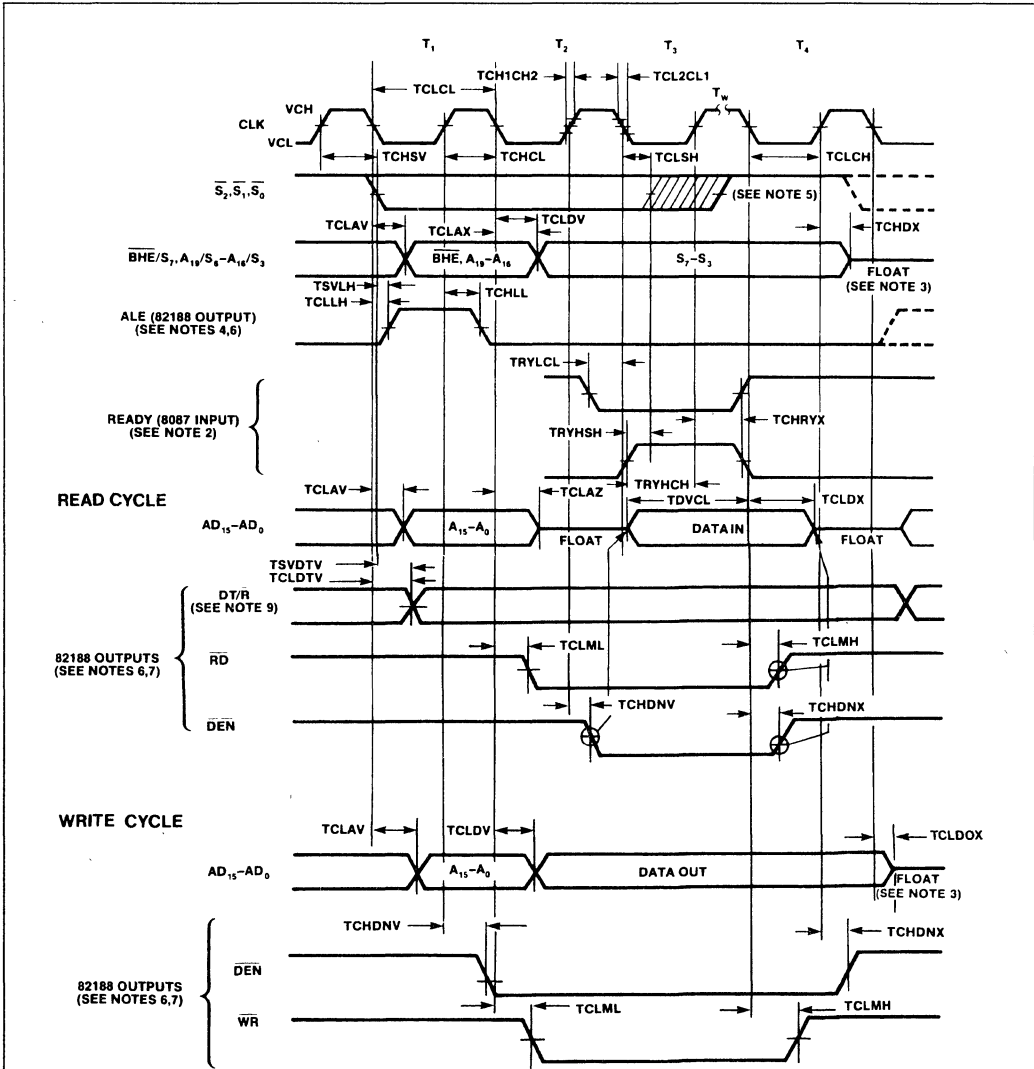
NOTES:

1. All signals switch between V_{OL} and V_{OH} unless otherwise specified.
2. READY is sampled near the end of T_2 , T_3 and T_W to determine if T_W machine states are to be inserted.
3. The local bus floats only if the 8087 is returning control to the 8086/8088.
4. ALE rises at later of (TSQLH, TCLLH).
5. Status inactive in state just prior to T_4 .
6. Signals at 8284A or 8288 are shown for reference only.
7. The issuance of 8288 command and control signals (\overline{MRDC} , \overline{MWTC} , \overline{AMWC} , and DEN) lags the active high 8288 CEN.
8. All timing measurements are made at 1.5V unless otherwise noted.

205835-10

WAVEFORMS (Continued)

MASTER MODE (with 82188 references)



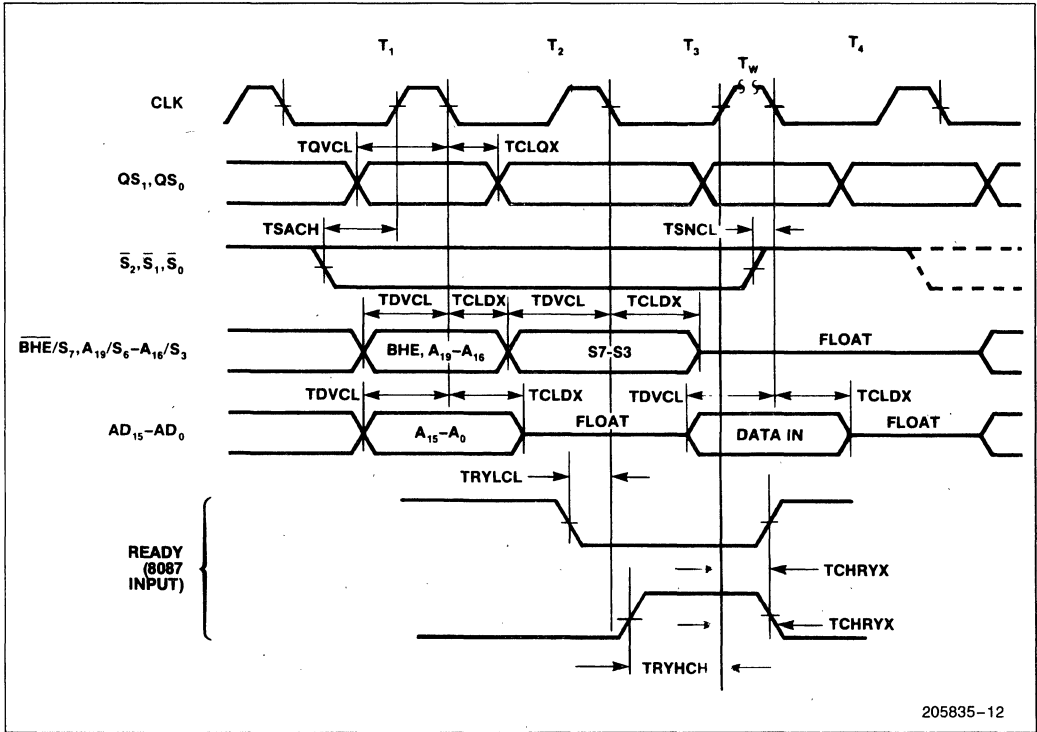
NOTES:

1. All signals switch between V_{OL} and V_{OH} unless otherwise specified.
2. READY is sampled near the end of T₂, T₃ and T_w to determine if T_w machine states are to be inserted.
3. The local bus floats only if the 8087 is returning control to the 80186/80188.
4. ALE rises at later of (T_{SVLH}, T_{CLLH}).
5. Status inactive in state just prior to T₄.
6. Signals at 8284A or 82188 are shown for reference only.
7. The issuance of 8288 command and control signals (\overline{MRDC} , \overline{AMWC} , and DEN) lags the active high 8288 CEN.
8. All timing measurements are made at 1.5V unless otherwise noted.
9. DT/R becomes valid at the later of (T_{SVDTV}, T_{TCLDTV}).

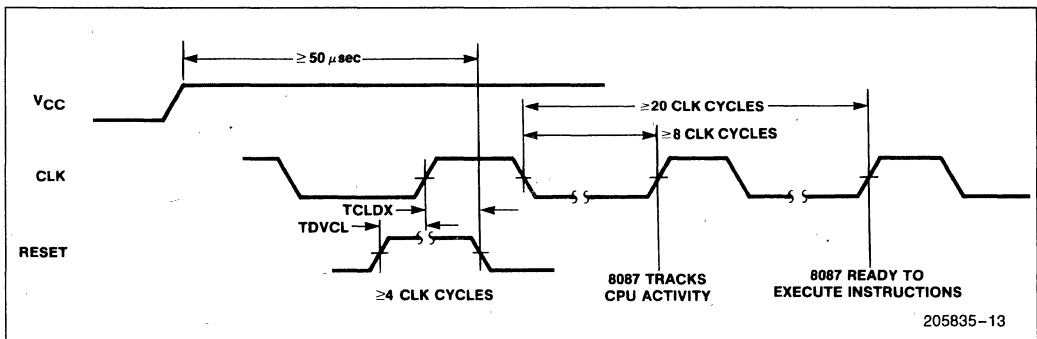
205835-11

WAVEFORMS (Continued)

PASSIVE MODE

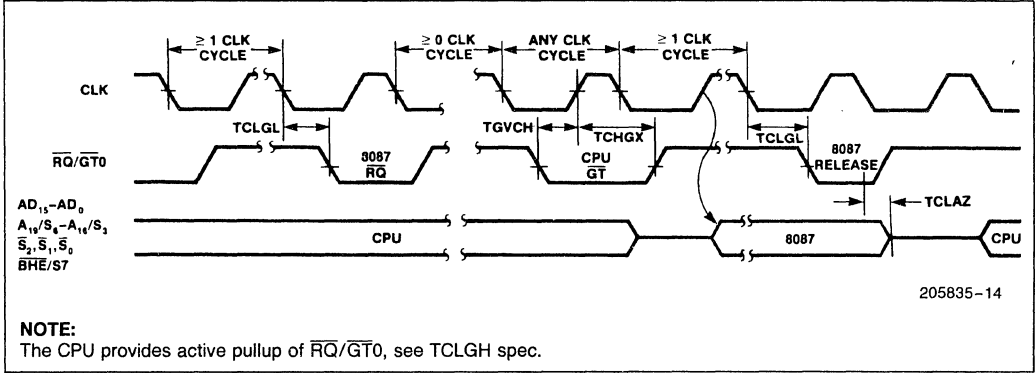


RESET TIMING

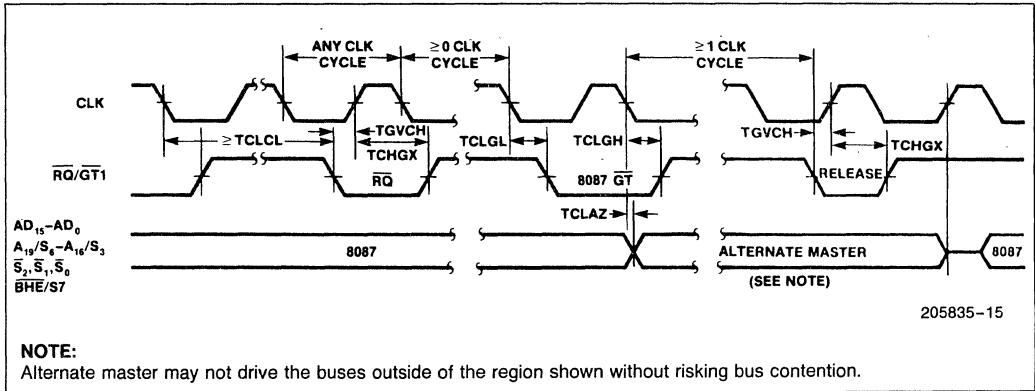


WAVEFORMS (Continued)

REQUEST/GRANT₀ TIMING



REQUEST/GRANT₁ TIMING



BUSY AND INTERRUPT TIMING

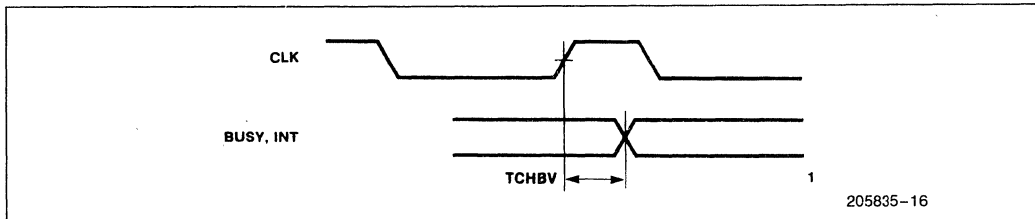


Table 5. 8087 Extensions to the 86/186 Instructions Sets

Data Transfer	Optional 8,16 Bit Displacement		Clock Count Range				
			32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer	
FLD = LOAD	MF =		00	01	10	11	
Integer/Real Memory to ST(0)	ESCAPE MF 1	MOD 0 0 0 R/M	DISP	38-56 +EA	52-60 +EA	40-60 +EA	46-54 +EA
Long Integer Memory to ST(0)	ESCAPE 1 1 1	MOD 1 0 1 R/M	DISP	60-68 +EA			
Temporary Real Memory to ST(0)	ESCAPE 0 1 1	MOD 1 0 1 R/M	DISP	53-65 +EA			
BCD Memory to ST(0)	ESCAPE 1 1 1	MOD 1 0 0 R/M	DISP	290-310 +EA			
ST(i) to ST(0)	ESCAPE 0 0 1	1 1 0 0 0 ST(i)		17-22			
FST = STORE							
ST(0) to Integer/Real Memory	ESCAPE MF 1	MOD 0 1 0 R/M	DISP	84-90 +EA	82-92 +EA	96-104 +EA	80-90 +EA
ST(0) to ST(i)	ESCAPE 1 0 1	1 1 0 1 0 ST(i)		15-22			
FSTP = STORE AND POP							
ST(0) to Integer/Real Memory	ESCAPE MF 1	MOD 0 1 1 R/M	DISP	86-92 +EA	84-94 +EA	98-106 +EA	82-92 +EA
ST(0) to Long Integer Memory	ESCAPE 1 1 1	MOD 1 1 1 R/M	DISP	94-105 +EA			
ST(0) to Temporary Real Memory	ESCAPE 0 1 1	MOD 1 1 1 R/M	DISP	52-58 +EA			
ST(0) to BCD Memory	ESCAPE 1 1 1	MOD 1 1 0 R/M	DISP	520-540 +EA			
ST(0) to ST(i)	ESCAPE 1 0 1	1 1 0 1 1 ST(i)		17-24			
FXCH = Exchange ST(i) and ST(0)	ESCAPE 0 0 1	1 1 0 0 1 ST(i)		10-15			
Comparison							
FCOM = Compare							
Integer/Real Memory to ST(0)	ESCAPE MF 0	MOD 0 1 0 R/M	DISP	60-70 +EA	78-91 +EA	65-75 +EA	72-86 +EA
ST(i) to ST(0)	ESCAPE 0 0 0	1 1 0 1 0 ST(i)		40-50			
FCOMP = Compare and Pop							
Integer/Real Memory to ST(0)	ESCAPE MF 0	MOD 0 1 1 R/M	DISP	63-73 +EA	80-93 +EA	67-77 +EA	74-88 +EA
ST(i) to ST(0)	ESCAPE 0 0 0	1 1 0 1 1 ST(i)		45-52			
FCOMPP = Compare ST(1) to ST(0) and Pop Twice	ESCAPE 1 1 0	1 1 0 1 1 0 0 1		45-55			
FTST = Test ST(0)	ESCAPE 0 0 1	1 1 1 0 0 1 0 0		38-48			
FXAM = Examine ST(0)	ESCAPE 0 0 1	1 1 1 0 0 1 0 1		12-23			

Table 5. 8087 Extensions to the 86/186 Instructions Sets (Continued)

Constants			Optional 8,16 Bit Displacement	Clock Count Range				
	MF	=		32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer	
				00	01	10	11	
FLDZ = LOAD + 0.0 into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 1 1 0				11-17	
FLD1 = LOAD + 1.0 into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 0 0 0				15-21	
FLDPI = LOAD π into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 0 1 1				16-22	
FLDL2T = LOAD $\log_2 10$ into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 0 0 1				16-22	
FLDL2E = LOAD $\log_2 e$ into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 0 1 0				15-21	
FLDLG2 = LOAD $\log_{10} 2$ into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 1 0 0				18-24	
FLDLN2 = LOAD $\log_e 2$ into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 1 0 1				17-23	
Arithmetic								
FADD = Addition								
Integer/Real Memory with ST(0)	ESCAPE	MF 0	MOD 0 0 0 R/M	DISP	90-120 + EA	108-143 + EA	95-125 + EA	102-137 + EA
ST(i) and ST(0)	ESCAPE	d P 0	1 1 0 0 0 ST(i)		70-100 (Note 1)			
FSUB = Subtraction								
Integer/Real Memory with ST(0)	ESCAPE	MF 0	MOD 1 0 R R/M	DISP	90-120 + EA	108-143 + EA	95-125 + EA	102-137 + EA
ST(i) and ST(0)	ESCAPE	d P 0	1 1 1 0 R R/M		70-100 (Note 1)			
FMUL = Multiplication								
Integer/Real Memory with ST(0)	ESCAPE	MF 0	MOD 0 0 1 R/M	DISP	110-125 + EA	130-144 + EA	112-168 + EA	124-138 + EA
ST(i) and ST(0)	ESCAPE	d P 0	1 1 0 0 1 R/M		90-145 (Note 1)			
FDIV = Division								
Integer/Real Memory with ST(0)	ESCAPE	MF 0	MOD 1 1 R R/M	DISP	215-225 + EA	230-243 + EA	220-230 + EA	224-238 + EA
ST(i) and ST(0)	ESCAPE	d P 0	1 1 1 1 R R/M		193-203 (Note 1)			
FSQRT = Square Root of ST(0)	ESCAPE	0 0 1	1 1 1 1 1 0 1 0				180-186	
FSCALE = Scale ST(0) by ST(1)	ESCAPE	0 0 1	1 1 1 1 1 1 0 1				32-38	
FPREM = Partial Remainder of ST(0) \div ST(1)	ESCAPE	0 0 1	1 1 1 1 1 0 0 0				15-190	
FRNDINT = Round ST(0) to Integer	ESCAPE	0 0 1	1 1 1 1 1 1 0 0				16-50	

205835-18

NOTE:

1. If P = 1 then add 5 clocks.

Table 5. 8087 Extensions to the 86/186 Instructions Sets (Continued)

		Optional 8,16 Bit Displacement	Clock Count Range
FTRACT - Extract Components of ST(0)	ESCAPE 0 0 1	1 1 1 1 0 1 0 0	27-55
FABS = Absolute Value of ST(0)	ESCAPE 0 0 1	1 1 1 0 0 0 0 1	10-17
FCHS = Change Sign of ST(0)	ESCAPE 0 0 1	1 1 1 0 0 0 0 0	10-17
Transcendental			
FPTAN = Partial Tangent of ST(0)	ESCAPE 0 0 1	1 1 1 1 0 0 1 0	30-540
FPATAN = Partial Arctangent of ST(0) - ST(1)	ESCAPE 0 0 1	1 1 1 1 0 0 1 1	250-800
F2XM1 = $2^{ST(0)} - 1$	ESCAPE 0 0 1	1 1 1 1 0 0 0 0	310-630
FYL2X = ST(1) • Log ₂ ST(0)	ESCAPE 0 0 1	1 1 1 1 0 0 0 1	900-1100
FYL2XP1 = ST(1) • Log ₂ ST(0) + 1	ESCAPE 0 0 1	1 1 1 1 1 0 0 1	700-1000
Processor Control			
FINIT = Initialized 8087	ESCAPE 0 1 1	1 1 1 0 0 0 1 1	2-8
FENI = Enable Interrupts	ESCAPE 0 1 1	1 1 1 0 0 0 0 0	2-8
FDISI = Disable Interrupts	ESCAPE 0 1 1	1 1 1 0 0 0 0 1	2-8
FLDCW = Load Control Word	ESCAPE 0 0 1	MOD 1 0 1 R/M	DISP 7-14 + EA
FSTCW = Store Control Word	ESCAPE 0 0 1	MOD 1 1 1 R/M	DISP 12-18 + EA
FSTSW = Store Status Word	ESCAPE 1 0 1	MOD 1 1 1 R/M	DISP 12-18 + EA
FCLEX = Clear Exceptions	ESCAPE 0 1 1	1 1 1 0 0 0 1 0	2-8
FSTENV = Store Environment	ESCAPE 0 0 1	MOD 1 1 0 R/M	DISP 40-50 + EA
FLDENV = Load Environment	ESCAPE 0 0 1	MOD 1 0 0 R/M	DISP 35-45 + EA
FAVE = Save State	ESCAPE 1 0 1	MOD 1 1 0 R/M	DISP 197-207 + EA
FRSTOR = Restore State	ESCAPE 1 0 1	MOD 1 0 0 R/M	DISP 197-207 + EA
FINCSTP = Increment Stack Pointer	ESCAPE 0 0 1	1 1 1 1 0 1 1 1	6-12
FDECSTP = Decrement Stack Pointer	ESCAPE 0 0 1	1 1 1 1 0 1 1 0	6-12

Table 5. 8087 Extensions to the 86/186 Instructions Sets (Continued)

		Clock Count Range
FFREE = Free ST(i)	ESCAPE 1 0 1 1 1 0 0 0 ST(i)	9-16
FNOP = No Operation	ESCAPE 0 0 1 1 1 0 1 0 0 0 0	10-16
FWAIT = CPU Wait for 8087	1 0 0 1 1 0 1 1	3+5n*
		205835-20

*n = number of times CPU examines TEST line before 8087 lowers BUSY.

NOTES:

1. if mod = 00 then DISP = 0*, disp-low and disp-high are absent
 if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
 if mod = 10 then DISP = disp-high; disp-low
 if mod = 11 then r/m is treated as an ST(i) field
2. if r/m = 000 then EA = (BX) + (SI) + DISP
 if r/m = 001 then EA = (BX) + (DI) + DISP
 if r/m = 010 then EA = (BP) + (SI) + DISP
 if r/m = 011 then EA = (BP) + (DI) + DISP
 if r/m = 100 then EA = (SI) + DISP
 if r/m = 101 then EA = (DI) + DISP
 if r/m = 110 then EA = (BP) + DISP
 if r/m = 111 then EA = (BX) + DISP
 *except if mod = 000 and r/m = 110 then EA = disp-high; disp-low.
3. MF = Memory Format
 00-32-bit Real
 01-32-bit Integer
 10-64-bit Real
 11-16-bit Integer
4. ST(0) = Current stack top
 ST(i) = ith register below stack top
5. d = Destination
 0-Destination is ST(0)
 1-Destination is ST(i)
6. P = Pop
 0-No pop
 1-Pop ST(0)
7. R = Reverse: When d = 1 reverse the sense of R
 0-Destination (op) Source
 1-Source (op) Destination
8. For **FSQRT**: $-0 \leq ST(0) \leq +\infty$
 For **FSCALE**: $-2^{15} \leq ST(1) < +2^{15}$ and ST(1) integer
 For **F2XM1**: $0 \leq ST(0) \leq 2^{-1}$
 For **FYL2X**: $0 < ST(0) < \infty$
 $-\infty < ST(1) < +\infty$
 For **FYL2XP1**: $0 \leq |ST(0)| < (2 - \sqrt{2})/2$
 $-\infty < ST(1) < \infty$
 For **FPTAN**: $0 \leq ST(0) \leq \pi/4$
 For **FPATAN**: $0 \leq ST(0) < ST(1) < +\infty$

80C286

HIGH PERFORMANCE CHMOS MICROPROCESSOR WITH MEMORY MANAGEMENT AND PROTECTION

- **High Speed CHMOS III Technology**

■ **Pin for Pin, Clock for Clock, and Functionally Compatible with the HMOS 80286**
(See 80286 Data Sheet, Order # 210253)

■ **Stop Clock Capability**
— **Uses Less Power (see I_{CCS} Specification)**
- **12.5 MHz Clock Rate**

■ **Available in a Variety of Packages:**
— **68 Pin PLCC (Plastic Leaded Chip Carrier)**
— **68 Pin PGA (Pin Grid Array)**
(See Packaging Spec., Order # 231369)

INTRODUCTION

The 80C286 is an advanced 16 bit CHMOS III microprocessor designed for multi-user and multi-tasking applications that require low power and high performance. The 80C286 is fully compatible with its predecessor the HMOS 80286 and object-code compatible with the 8086 and 80386 family of products. In addition, the 80C286 has a power down mode which uses less power, making it ideal for mobile applications. The 80C286 has built-in memory protection that maintains a four level protection mechanism for task isolation, a hardware task switching facility and memory management capabilities that map 2³⁰ bytes (one gigabyte) of virtual address space per task (per user) into 2²⁴ bytes (16 megabytes) of physical memory.

The 80C286 is upward compatible with 8086 and 8088 software. Using 8086 real address mode, the 80C286 is object code compatible with existing 8086, 8088 software. In protected virtual address mode, the 80C286 is source code compatible with 8086, 8088 software which may require upgrading to use virtual addresses supported by the 80C286's integrated memory management and protection mechanism. Both modes operate at full 80C286 performance and execute a superset of the 8086 and 8088 instructions.

The 80C286 provides special operations to support the efficient implementation and execution of operating systems. For example, one instruction can end execution of one task, save its state, switch to a new task, load its state, and start execution of the new task. The 80C286 also supports virtual memory systems by providing a segment-not-present exception and restartable instructions.

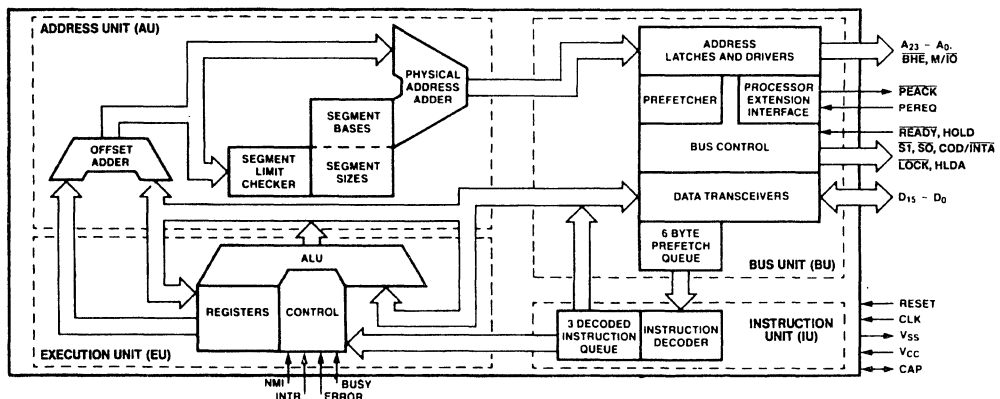
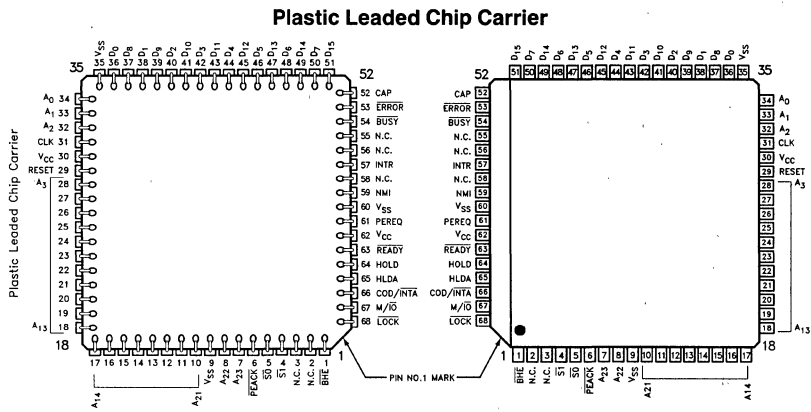


Figure 1. 80C286 Internal Block Diagram

231923-1

Component Pad Views—As viewed from underside of component when mounted on the board.

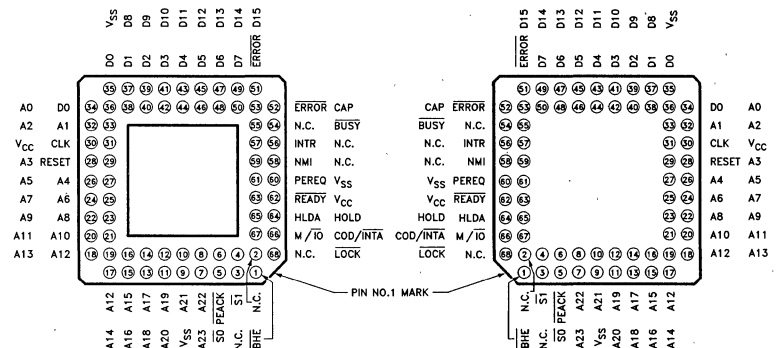
P.C. Board Views—As viewed from the component side of the P.C. board.



231923-2

NOTE:
N.C. signals must not be connected

Pin Grid Array



231923-3

Figure 2. 80C286 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for the 80C286 microprocessor:

Symbol	Type	Name and Function
CLK	I	SYSTEM CLOCK provides the fundamental timing for 80C286 systems. It is divided by two inside the 80C286 to generate the processor clock. The internal divide-by-two circuitry can be synchronized to an external clock generator by a LOW to HIGH transition on the RESET input.
D ₁₅ -D ₀	I/O	DATA BUS inputs data during memory, I/O, and interrupt acknowledge read cycles; outputs data during memory and I/O write cycles. The data bus is active HIGH and floats to 3-state OFF* during bus hold acknowledge.
A ₂₃ -A ₀	O	ADDRESS BUS outputs physical memory and I/O port addresses. A0 is LOW when data is to be transferred on pins D ₇ -D ₀ . A ₂₃ -A ₁₆ are LOW during I/O transfers. The address bus is active HIGH and floats to 3-state OFF* during bus hold acknowledge.
BHE	O	BUS HIGH ENABLE indicates transfer of data on the upper byte of the data bus. D ₁₅ -D ₈ . Eight-bit oriented devices assigned to the upper byte of the data bus would normally use BHE to condition chip select functions. BHE is active LOW and floats to 3-state OFF* during bus hold acknowledge.

*See bus hold circuitry section.

Table I. Pin Description (Continued)

Symbol	Type	Name and Function				
BHE (Continued)		BHE and A0 Encodings				
		BHE Value	A0 Value	Function		
		0	0	Word transfer		
		0	1	Transfer on upper half of data bus (D ₁₅ -D ₈)		
		1	0	Byte transfer on lower half of data bus (D ₇ -D ₀)		
		1	1	Will never occur		
S ₁ , S ₀	O	BUS CYCLE STATUS indicates initiation of a bus cycle and, along with M/ \overline{IO} and COD/ \overline{INTA} , defines the type of bus cycle. The bus is in a T _S state whenever one or both are LOW, S ₁ and S ₀ are active LOW and float to 3-state OFF* during bus hold acknowledge.				
		80C286 Bus Cycle Status Definition				
		COD/\overline{INTA}	M/\overline{IO}	S₁	S₀	Bus Cycle Initiated
		0 (LOW)	0	0	0	Interrupt acknowledge
		0	0	0	1	Will not occur
		0	0	1	0	Will not occur
		0	0	1	1	None; not a status cycle
		0	1	0	0	IF A ₁ = 1 then halt; else shutdown
		0	1	0	1	Memory data read
		0	1	1	0	Memory data write
0	1	1	1	None; not a status cycle		
1 (HIGH)	0	0	0	Will not occur		
1	0	0	1	I/O read		
1	0	1	0	I/O write		
1	0	1	1	None; not a status cycle		
1	1	0	0	Will not occur		
1	1	0	1	Memory instruction read		
1	1	1	0	Will not occur		
1	1	1	1	None; not a status cycle		
M/ \overline{IO}	O	MEMORY I/O SELECT distinguishes memory access from I/O access. If HIGH during T _S , a memory cycle or a halt/shutdown cycle is in progress. If LOW, an I/O cycle or an interrupt acknowledge cycle is in progress. M/ \overline{IO} floats to 3-state OFF* during bus hold acknowledge.				
COD/ \overline{INTA}	O	CODE/INTERRUPT ACKNOWLEDGE distinguishes instruction fetch cycles from memory data read cycles. Also distinguishes interrupt acknowledge cycles from I/O cycles. COD/ \overline{INTA} floats to 3-state OFF* during bus hold acknowledge. Its timing is the same as M/ \overline{IO} .				
LOCK	O	BUS LOCK indicates that other system bus masters are not to gain control of the system bus for the current and the following bus cycle. The LOCK signal may be activated explicitly by the "LOCK" instruction prefix or automatically by 80C286 hardware during memory XCHG instructions, interrupt acknowledge, or descriptor table access. LOCK is active LOW and floats to 3-state OFF* during bus hold acknowledge.				
READY	I	BUS READY terminates a bus cycle. Bus cycles are extended without limit until terminated by READY LOW. READY is an active LOW synchronous input requiring setup and hold times relative to the system clock be met for correct operation. READY is ignored during bus hold acknowledge.				
HOLD HLDA	I O	BUS HOLD REQUEST AND HOLD ACKNOWLEDGE control ownership of the 80C286 local bus. The HOLD input allows another local bus master to request control of the local bus. When control is granted, the 80C286 will float its bus drivers to 3-state OFF* and then activate HLDA, thus entering the bus hold acknowledge condition. The local bus will remain granted to the requesting master until HOLD becomes inactive which results in the 80C286 deactivating HLDA and regaining control of the local bus. This terminates the bus hold acknowledge condition. HOLD may be asynchronous to the system clock. These signals are active HIGH.				
INTR	I	INTERRUPT REQUEST requests the 80C286 to suspend its current program execution and service a pending external request. Interrupt requests are masked whenever the interrupt enable bit in the flag word is cleared. When the 80C286 responds to an interrupt request, it performs two interrupt acknowledge bus cycles to read an 8-bit interrupt vector that identifies the source of the interrupt. To assure program interruption, INTR must remain active until the first interrupt acknowledge cycle is completed. INTR is sampled at the beginning of each processor cycle and must be active HIGH at least two processor cycles before the current instruction ends in order to interrupt before the next instruction. INTR is level sensitive, active HIGH, and may be asynchronous to the system clock.				

*See bus hold circuitry section.

Table 1. Pin Description (Continued)

Symbol	Type	Name and Function	
NMI	I	NON-MASKABLE INTERRUPT REQUEST interrupts the 80C286 with an internally supplied vector value of 2. No interrupt acknowledge cycles are performed. The interrupt enable bit in the 80C286 flag word does not affect this input. The NMI input is active HIGH, may be asynchronous to the system clock, and is edge triggered after internal synchronization. For proper recognition, the input must have been previously LOW for at least four system clock cycles and remain HIGH for at least four system clock cycles.	
PEREQ PEACK	I O	PROCESSOR EXTENSION OPERAND REQUEST AND ACKNOWLEDGE extend the memory management and protection capabilities of the 80C286 to processor extensions. The PEREQ input requests the 80C286 to perform a data operand transfer for a processor extension. The PEACK output signals the processor extension when the requested operand is being transferred. PEREQ is active HIGH and floats to 3-state OFF* during bus hold. PEACK is active LOW.	
BUSY ERROR	I I	PROCESSOR EXTENSION BUSY AND ERROR indicate the operating condition of a processor extension to the 80C286. An active BUSY input stops 80C286 program execution on WAIT and some ESC instructions until BUSY becomes inactive (HIGH). The 80C286 may be interrupted while waiting for BUSY to become inactive. An active ERROR input causes the 80C286 to perform a processor extension interrupt when executing WAIT or some ESC instructions. These inputs are active LOW and may be asynchronous to the system clock. These inputs have internal pull-up resistors.	
RESET	I	SYSTEM RESET clears the internal logic of the 80C286 and is active HIGH. The 80C286 may be reinitialized at any time with a LOW to HIGH transition on RESET which remains active for more than 16 system clock cycles. During RESET active, the output pins of the 80C286 enter the state shown below:	
		80C286 Pin State During Reset	
		Pin Value	Pin Names
		1 (HIGH) 0 (LOW) 3-state OFF*	S0, ST, PEACK, A23-A0, BHE, LOCK M/I0, COD/INTA, HLDA (Note 1) D15-D0
		Operation of the 80C286 begins after a HIGH to LOW transition on RESET. The HIGH to LOW transition of RESET must be synchronous to the system clock. Approximately 38 CLK cycles from the trailing edge of RESET are required by the 80C286 for internal initialization before the first bus cycle, to fetch code from the power-on execution address, occurs. A LOW to HIGH transition of RESET synchronous to the system clock will end a processor cycle at the second HIGH to LOW transition of the system clock. The LOW to HIGH transition of RESET may be asynchronous to the system clock; however, in this case it cannot be predetermined which phase of the processor clock will occur during the next system clock period. Synchronous LOW to HIGH transitions of RESET are required only for systems where the processor clock must be phase synchronous to another clock.	
V _{SS}	I	SYSTEM GROUND: 0 Volts.	
V _{CC}	I	SYSTEM POWER: +5 Volt Power Supply.	
CAP	I	SUBSTRATE FILTER CAPACITOR: a 0.047 μ F \pm 20% 12V capacitor can be connected between this pin and ground for compatibility with the HMOS 80286. For systems using only an 80C286, this pin can be left floating.	

*See bus hold circuitry section.

NOTE:

1. HLDA is only Low if HOLD is inactive (Low).

FUNCTIONAL DESCRIPTION

Introduction

The 80C286 is an advanced, high-performance microprocessor with specially optimized capabilities for multiple user and multi-tasking systems. Depending on the application, a 12 MHz 80C286's performance is up to ten times faster than the standard 5 MHz 8086's, while providing complete upward software compatibility with Intel's 8086, 88, and 186 family of CPU's.

The 80C286 operates in two modes: 8086 real address mode and protected virtual address mode. Both modes execute a superset of the 8086 and 88 instruction set.

In 8086 real address mode programs use real addresses with up to one megabyte of address space. Programs use virtual addresses in protected virtual address mode, also called protected mode. In protected mode, the 80C286 CPU automatically maps 1 gigabyte of virtual addresses per task into a 16 megabyte real address space. This mode also provides memory protection to isolate the operating system and ensure privacy of each tasks' programs and data. Both modes provide the same base instruction set, registers, and addressing modes.

The following Functional Description describes first, the base 80C286 architecture common to both modes, second, 8086 real address mode, and third, protected mode.

80C286 BASE ARCHITECTURE

The 8086, 88, 186, and 286 CPU family all contain the same basic set of registers, instructions, and

addressing modes. The 80C286 processor is upward compatible with the 8086, 8088, and 80186 CPU's and fully compatible with the HMOS 80286.

Register Set

The 80C286 base architecture has fifteen registers as shown in Figure 3. These registers are grouped into the following four categories:

General Registers: Eight 16-bit general purpose registers used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used either in their entirety as 16-bit words or split into pairs of separate 8-bit registers.

Segment Registers: Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers: Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode determines the specific registers used for operand address calculations.

Status and Control Registers: The 3 16-bit special purpose registers in figure 3A record or control certain aspects of the 80C286 processor state including the Instruction Pointer, which contains the offset address of the next sequential instruction to be executed.

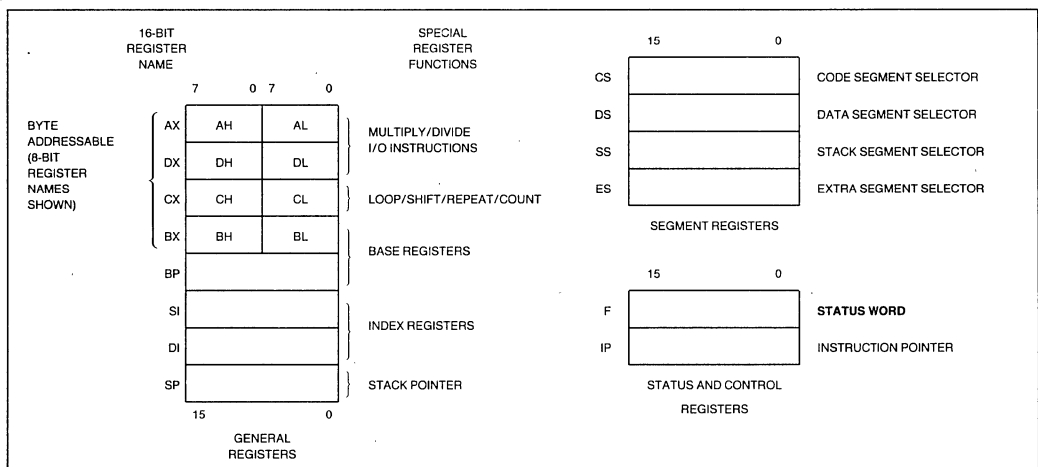


Figure 3. Register Set

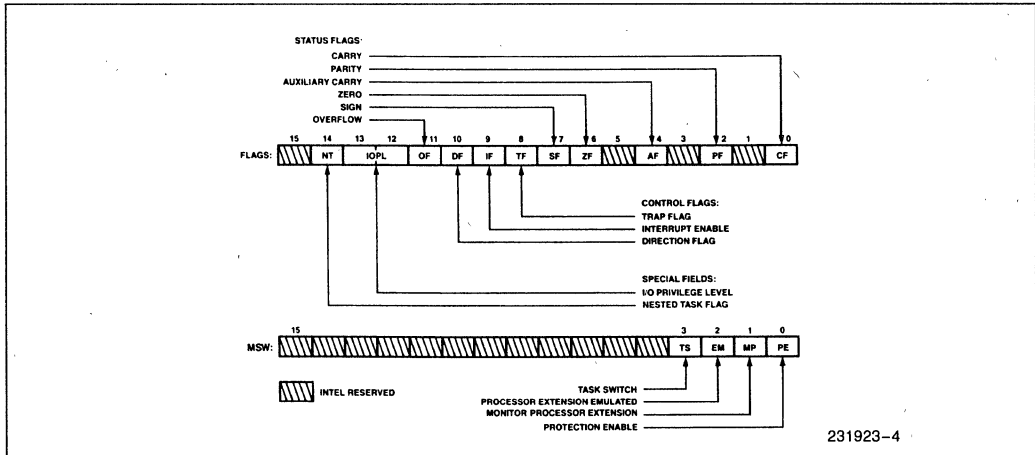


Figure 3a. Status and Control Register Bit Functions

Flags Word Description

The Flags word (Flags) records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80C286 within a given operating mode (bits 8 and 9). Flags is a 16-bit register. The function of the flag bits is given in Table 2.

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high level instructions, and processor control. These categories are summarized in Figure 4.

An 80C286 instruction can reference zero, one, or two operands; where an operand resides in a register, in the instruction itself, or in memory. Zero-operand instructions (e.g. NOP and HLT) are usually one byte long. One-operand instructions (e.g. INC and DEC) are usually two bytes long but some are encoded in only one byte. One-operand instructions may reference a register or memory location. Two-operand instructions permit the following six types of instruction operations:

- Register to Register
- Memory to Register
- Immediate to Register
- Memory to Memory
- Register to Memory
- Immediate to Memory

Table 2. Flags Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
11	OF	Overflow Flag—Set if result is a too-large positive number or a too-small negative number (excluding sign-bit) to fit in destination operand; cleared otherwise
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index registers when set. Clearing DF causes auto increment.

Two-operand instructions (e.g. MOV and ADD) are usually three to six bytes long. Memory to memory operations are provided by a special class of string instructions requiring one to three bytes. For detailed instruction formats and encodings refer to the instruction set summary at the end of this document.

For detailed operation and usage of each instruction, see Appendix B of the 80286/80287 Programmer's Reference Manual (Order No. 210498).

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack

Figure 4a. Data Transfer Instructions

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero

Figure 4c. String Instructions

ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiple byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

Figure 4b. Arithmetic Instructions

LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

Figure 4d. Shift/Rotate Logical Instructions

CONDITIONAL TRANSFERS		UNCONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal	CALL	Call procedure
JAE/JNB	Jump if above or equal/not below	RET	Return from procedure
JB/JNAE	Jump if below/not above nor equal	JMP	Jump
JBE/JNA	Jump if below or equal/not above	ITERATION CONTROLS	
JC	Jump if carry		
JE/JZ	Jump if equal/zero		
JG/JNLE	Jump if greater/not less nor equal		
JGE/JNL	Jump if greater or equal/not less	LOOP	Loop
JL/JNGE	Jump if less/not greater nor equal	LOOPE/LOOPZ	Loop if equal/zero
JLE/JNG	Jump if less or equal/not greater	LOOPNE/LOOPNZ	Loop if not equal/not zero
JNC	Jump if not carry	JCXZ	Jump if register CX = 0
JNE/JNZ	Jump if not equal/not zero	INTERRUPTS	
JNO	Jump if not overflow		
JNP/JPO	Jump if not parity/parity odd	INT	Interrupt
JNS	Jump if not sign	INTO	Interrupt if overflow
JO	Jump if overflow	IRET	Interrupt return
JP/JPE	Jump if parity/parity even		
JS	Jump if sign		

Figure 4e. Program Transfer Instructions

FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for BUSY not active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
EXECUTION ENVIRONMENT CONTROL	
LMSW	Load machine status word
SMSW	Store machine status word

Figure 4f. Processor Control Instructions

ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4g. High Level Instructions

Memory Organization

Memory is organized as sets of variable length segments. Each segment is a linear contiguous sequence of up to 64K (2¹⁶) 8-bit bytes. Memory is addressed using a two component address (a pointer) that consists of a 16-bit segment selector, and a 16-bit offset. The segment selector indicates the desired segment in memory. The offset component indicates the desired byte address within the segment.

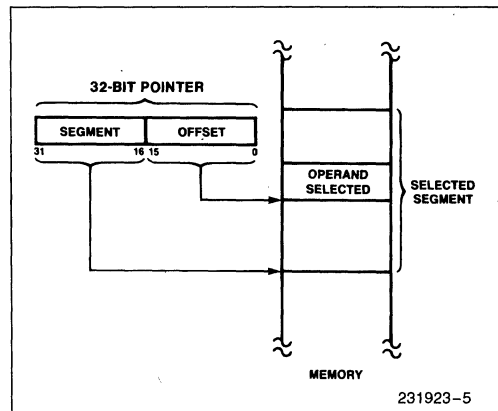


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Automatic with instruction prefetch
Stack	Stack (SS)	All stack pushes and pops. Any memory reference which uses BP as a base register.
Local Data	Data (DS)	All data references except when relative to stack or string destination
External (Global) Data	Extra (ES)	Alternate data segment and destination of string operation

All instructions that address operands in memory must specify the segment and the offset. For speed and compact instruction encoding, segment selectors are usually stored in the high speed segment registers. An instruction need specify only the desired segment register and an offset in order to address a memory operand.

Most instructions need not explicitly specify which segment register is used. The correct segment register is automatically chosen according to the rules of Table 3. These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs. To access operands not residing in one of the four immediately available segments, a full 32-bit pointer or a new segment selector must be loaded.

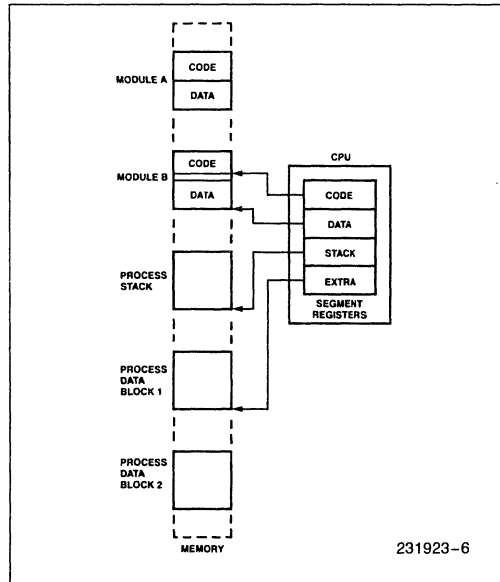


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80C286 provides a total of eight addressing modes for instructions to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8 or 16-bit general registers.

Immediate Operand Mode: The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: segment selector and offset. The segment selector is supplied by a segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset is calculated by summing any combination of the following three address elements:

the **displacement** (an 8 or 16-bit immediate value contained in the instruction)

the **base** (contents of either the BX or BP base registers)

the **index** (contents of either the SI or DI index registers)

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

Direct Mode: The operand's offset is contained in the instruction as an 8 or 16-bit displacement element.

Register Indirect Mode: The operand's offset is in one of the registers SI, DI, BX, or BP.

Based Mode: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of a base register (BX or BP).

Indexed Mode: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of an index register (SI or DI).

Based Indexed Mode: The operand's offset is the sum of the contents of a base register and an index register.

Based Indexed Mode with Displacement: The operand's offset is the sum of a base register's contents, an index register's contents, and an 8 or 16-bit displacement.

Data Types

The 80C286 directly supports the following data types:

- Integer:** A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32 and 64-bit integers are supported using the Numeric Data Processor, the 80287.
- Ordinal:** An unsigned binary numeric value contained in an 8-bit byte or 16-bit word.
- Pointer:** A 32-bit quantity, composed of a segment selector component and an offset component. Each component is a 16-bit word.
- String:** A contiguous sequence of bytes or words. A string may contain from 1 byte to 64K bytes.
- ASCII:** A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- BCD:** A byte (unpacked) representation of the decimal digits 0–9.
- Packed BCD:** A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble of the byte.
- Floating Point:** A signed 32, 64, or 80-bit real number representation. (Floating point operands are supported using the 80287 Numeric Processor).

Figure 7 graphically represents the data types supported by the 80C286.

either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A₁₅–A₈ are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

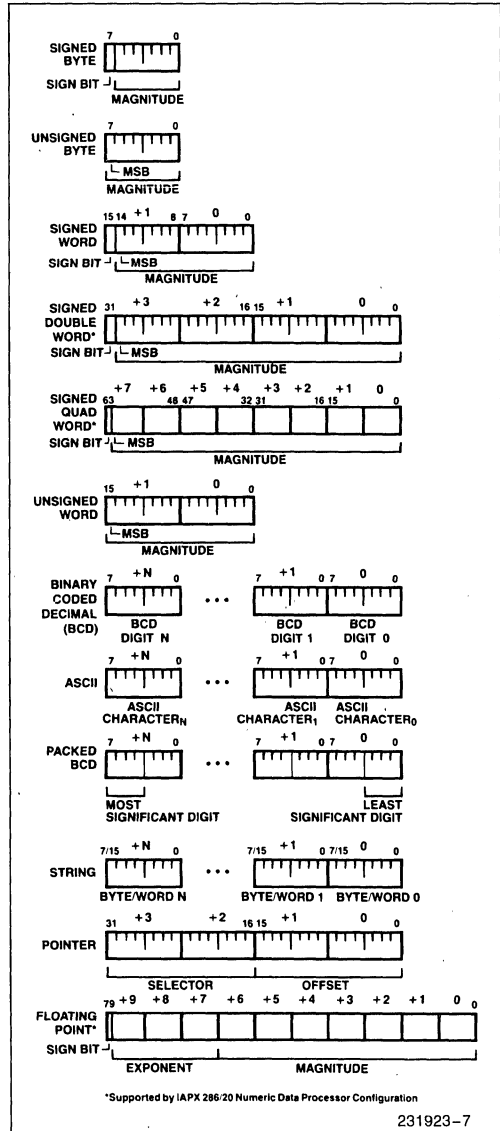


Figure 7. 80C286 Supported Data Types

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. I/O instructions address the I/O space with

Table 4. Interrupt Vector Assignments

Function	Interrupt Number	Related Instructions	Does Return Address Point to Instruction Causing Exception?
Divide error exception	0	DIV, IDIV	Yes
Single step interrupt	1	All	
NMI interrupt	2	INT 2 or NMI pin	
Breakpoint interrupt	3	INT 3	
INTO detected overflow exception	4	INTO	No
BOUND range exceeded exception	5	BOUND	Yes
Invalid opcode exception	6	Any undefined opcode	Yes
Processor extension not available exception	7	ESC or WAIT	Yes
Intel reserved—do not use	8-15		
Processor extension error interrupt	16	ESC or WAIT	
Intel reserved—do not use	17-31		
User defined	32-255		

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Flags) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable. Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. The return address from an exception will always point at the instruction causing the exception and include any leading instruction prefixes.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. For each interrupt, an 8-bit vector must be supplied to the 80C286 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

MASKABLE INTERRUPT (INTR)

The 80C286 provides a maskable hardware interrupt request pin, INTR. Software enables this input by

setting the interrupt flag bit (IF) in the flag word. All 224 user-defined interrupt sources can share this input, yet they can retain separate interrupt handlers. An 8-bit vector read by the CPU during the interrupt acknowledge sequence (discussed in System Interface section) identifies the source of the interrupt.

Further maskable interrupts are disabled while servicing an interrupt by resetting the IF but as part of the response to an interrupt or exception. The saved flag word will reflect the enable status of the processor prior to the interrupt. Until the flag word is restored to the flag register, the interrupt flag will be zero unless specifically set. The interrupt return instruction includes restoring the flag word, thereby restoring the original status of IF.

NON-MASKABLE INTERRUPT REQUEST (NMI)

A non-maskable interrupt input (NMI) is also provided. NMI has higher priority than INTR. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed.

While executing the NMI servicing procedure, the 80C286 will service neither further NMI requests, INTR requests, nor the processor extension segment overrun interrupt until an interrupt return (IRET) instruction is executed or the CPU is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. IF is cleared at the beginning of an NMI interrupt to inhibit INTR interrupts.

SINGLE STEP INTERRUPT

The 80C286 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single step interrupt and is controlled by the single step flag bit (TF) in the flag word. Once this bit is set, an internal single step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single stepped.

Interrupt Priorities

When simultaneous interrupt requests occur, they are processed in a fixed order as shown in Table 5. Interrupt processing involves saving the flags, return address, and setting CS:IP to point at the first instruction of the interrupt handler. If other interrupts remain enabled they are processed before the first instruction of the current interrupt handler is executed. The last interrupt processed is therefore the first one serviced.

Table 5. Interrupt Processing Order

Order	Interrupt
1	Instruction exception
2	Single step
3	NMI
4	Processor extension segment overrun
5	INTR
6	INT instruction

Initialization and Processor Reset

Processor initialization or start up is accomplished by driving the RESET input pin HIGH. RESET forces the 80C286 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active. After RESET becomes inactive and an internal processing interval elapses, the 80C286 begins execution in real address mode with the instruction at physical location FFFFFFF0(H). RESET also sets some registers to predefined values as shown in Table 6.

Table 6. 80C286 Initial Register State after RESET

Flag word	0002(H)
Machine Status Word	FFF0(H)
Instruction pointer	FFF0(H)
Code segment	F000(H)
Data segment	0000(H)
Extra segment	0000(H)
Stack segment	0000(H)

HOLD must not be active during the time from the leading edge of RESET to 34 CLKs after the trailing edge of RESET.

Machine Status Word Description

The machine status word (MSW) records when a task switch takes place and controls the operating mode of the 80C286. It is a 16-bit register of which the lower four bits are used. One bit places the CPU into protected mode, while the other three bits, as shown in Table 7, control the processor extension interface. After RESET, this register contains FFF0(H) which places the 80C286 in 8086 real address mode.

Table 7. MSW Bit Functions

Bit Position	Name	Function
0	PE	Protected mode enable places the 80C286 into protected mode and cannot be cleared except by RESET.
1	MP	Monitor processor extension allows WAIT instructions to cause a processor extension not present exception (number 7).
2	EM	Emulate processor extension causes a processor extension not present exception (number 7) on ESC instructions to allow emulating a processor extension.
3	TS	Task switched indicates the next instruction using a processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task.

The LMSW and SMSW instructions can load and store the MSW in real address mode. The recommended use of TS, EM, and MP is shown in Table 8.

Table 8. Recommended MSW Encodings For Processor Extension Control

TS	MP	EM	Recommended Use	Instructions Causing Exception 7
0	0	0	Initial encoding after RESET. 80C286 operation is identical to 8086, 88.	None
0	0	1	No processor extension is available. Software will emulate its function.	ESC
1	0	1	No processor extension is available. Software will emulate its function. The current processor extension context may belong to another task.	ESC
0	1	0	A processor extension exists.	None
1	1	0	A processor extension exists. The current processor extension context may belong to another task. The Exception 7 on WAIT allows software to test for an error pending from a previous processor extension operation.	ESC or WAIT

Halt

The HLT instruction stops program execution and prevents the CPU from using the local bus until restarted. Either NMI, INTR with IF = 1, or RESET will force the 80C286 out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

8086 REAL ADDRESS MODE

The 80C286 executes a fully upward-compatible superset of the 8086 instruction set in real address mode. In real address mode the 80C286 is object code compatible with 8086 and 8088 software. The real address mode architecture (registers and addressing modes) is exactly as described in the 80C286 Base Architecture section of this Functional Description.

Memory Size

Physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pins A₀ through A₁₉ and BHE. A₂₀ through A₂₃ should be ignored.

Memory Addressing

In real address mode physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pins A₀ through A₁₉ and BHE. Address bits A₂₀-A₂₃ may not always be zero in real mode. A₂₀-A₂₃ should not be used by the system while the 80C286 is operating in Real Mode.

The selector portion of a pointer is interpreted as the upper 16 bits of a 20-bit segment address. The lower four bits of the 20-bit segment address are always zero. Segment addresses, therefore, begin on multiples of 16 bytes. See Figure 8 for a graphic representation of address information.

All segments in real address mode are 64K bytes in size and may be read, written, or executed. An exception or interrupt can occur if data operands or instructions attempt to wrap around the end of a segment (e.g. a word with its low order byte at offset FFFF(H) and its high order byte at offset 0000(H)). If, in real address mode, the information contained in a segment does not use the full 64K bytes, the unused end of the segment may be overlaid by another segment to reduce physical memory requirements.

Reserved Memory Locations

The 80C286 reserves two fixed areas of memory in real address mode (see Figure 9); system initializa-

tion area and interrupt table area. Locations from addresses FFFF0(H) through FFFFF(H) are reserved for system initialization. Initial execution begins at location FFFF0(H). Locations 00000(H) through 003FF(H) are reserved for interrupt vectors.

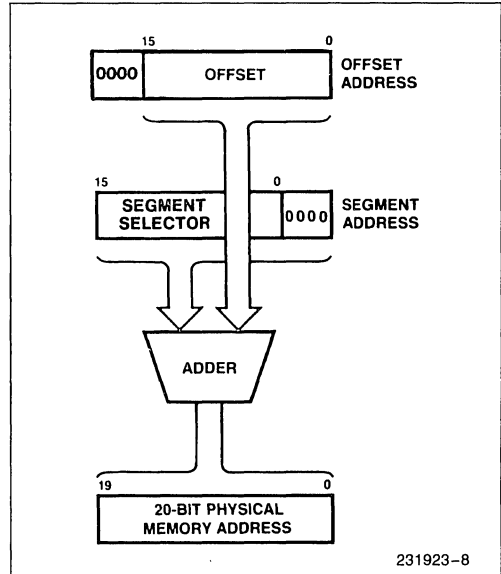


Figure 8. 8086 Real Address Mode Address Calculation

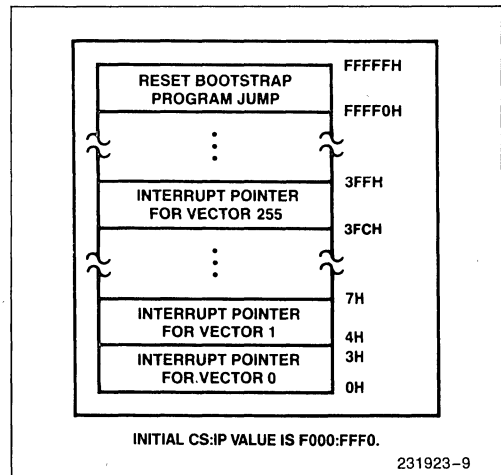


Figure 9. 8086 Real Address Mode Initially Reserved Memory Locations

Table 9. Real Address Mode Addressing Interrupts

Function	Interrupt Number	Related Instructions	Return Address Before Instruction?
Interrupt table limit too small exception	8	INT vector is not within table limit	Yes
Processor extension segment overrun interrupt	9	ESC with memory operand extending beyond offset FFFF(H)	No
Segment overrun exception	13	Word memory reference with offset = FFFF(H) or an attempt to execute past the end of a segment	Yes

Interrupts

Table 9 shows the interrupt vectors reserved for exceptions and interrupts which indicate an addressing error. The exceptions leave the CPU in the state existing before attempting to execute the failing instruction (except for PUSH, POP, PUSHA, or POPA). Refer to the next section on protected mode initialization for a discussion on exception 8.

Protected Mode Initialization

To prepare the 80C286 for protected mode, the LIDT instruction is used to load the 24-bit interrupt table base and 16-bit limit for the protected mode interrupt table. This instruction can also set a base and limit for the interrupt vector table in real address mode. After reset, the interrupt table base is initialized to 000000(H) and its size set to 03FF(H). These values are compatible with 8086, 88 software. LIDT should only be executed in preparation for protected mode.

Shutdown

Shutdown occurs when a severe error is detected that prevents further instruction processing by the CPU. Shutdown and halt are externally signalled via a halt bus operation. They can be distinguished by A_1 HIGH for halt and A_1 LOW for shutdown. In real address mode, shutdown can occur under two conditions:

- Exceptions 8 or 13 happen and the IDT limit does not include the interrupt vector.
- A CALL INT or PUSH instruction attempts to wrap around the stack segment when SP is not even.

An NMI input can bring the CPU out of shutdown if the IDT limit is at least 000F(H) and SP is greater than 0005(H), otherwise shutdown can only be exited via the RESET input.

PROTECTED VIRTUAL ADDRESS MODE

The 80C286 executes a fully upward-compatible superset of the 8086 instruction set in protected virtual address mode (protected mode). Protected mode also provides memory management and protection mechanisms and associated instructions.

The 80C286 enters protected virtual address mode from real address mode by setting the PE (Protection Enable) bit of the machine status word with the Load Machine Status Word (LMSW) instruction. Protected mode offers extended physical and virtual memory address space, memory protection mechanisms, and new operations to support operating systems and virtual memory.

All registers, instructions, and addressing modes described in the 80C286 Base Architecture section of this Functional Description remain the same. Programs for the 8086, 88, 186, and real address mode 80C286 can be run in protected mode; however, embedded constants for segment selectors are different.

Memory Size

The protected mode 80C286 provides a 1 gigabyte virtual address space per task mapped into a 16 megabyte physical address space defined by the address pin $A_{23..A_0}$ and \overline{BHE} . The virtual address space may be larger than the physical address space since any use of an address that does not map to a physical memory location will cause a restartable exception.

Memory Addressing

As in real address mode, protected mode uses 32-bit pointers, consisting of 16-bit selector and offset components. The selector, however, specifies an index into a memory resident table rather than the upper 16-bits of a real memory address. The 24-bit

base address of the desired segment is obtained from the tables in memory. The 16-bit offset is added to the segment base address to form the physical address as shown in Figure 10. The tables are automatically referenced by the CPU whenever a segment register is loaded with a selector. All 80C286 instructions which load a segment register will reference the memory based tables without additional software. The memory based tables contain 8 byte values called descriptors.

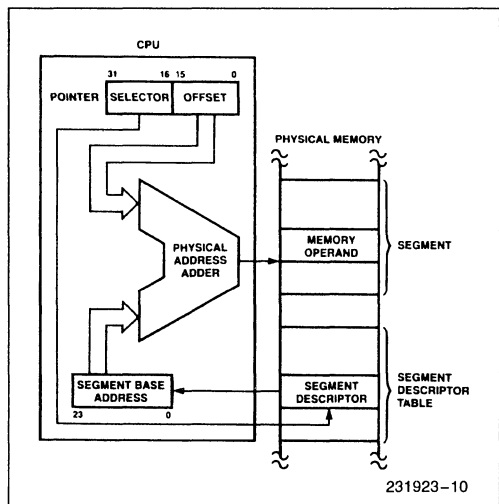


Figure 10. Protected Mode Memory Addressing

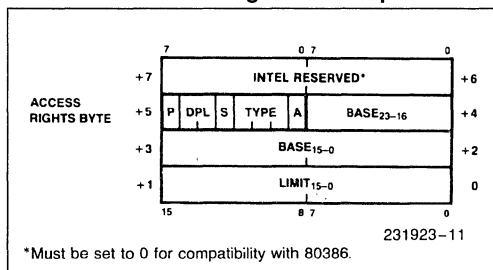
DESCRIPTORS

Descriptors define the use of memory. Special types of descriptors also define new functions for transfer of control and task switching. The 80C286 has segment descriptors for code, stack and data segments, and system control descriptors for special system data segments and control transfer operations. Descriptor accesses are performed as locked bus operations to assure descriptor integrity in multi-processor systems.

CODE AND DATA SEGMENT DESCRIPTORS (S = 1)

Besides segment base addresses, code and data descriptors contain other segment attributes including segment size (1 to 64K bytes), access rights (read only, read/write, execute only, and execute/read), and presence in memory (for virtual memory systems) (See Figure 11). Any segment usage violating a segment attribute indicated by the segment descriptor will prevent the memory cycle and cause an exception or interrupt.

Code or Data Segment Descriptor



Access Rights Byte Definition

Bit Position	Name	Function
7	Present (P)	P = 1 Segment is mapped into physical memory. P = 0 No mapping to physical memory exists, base and limit are not used.
6-5	Descriptor Privilege Level (DPL)	Segment privilege attribute used in privilege tests.
4	Segment Descriptor (S)	S = 1 Code or Data (includes stacks) segment descriptor S = 0 System Segment Descriptor or Gate Descriptor
3	Executable (E)	Data segment descriptor type is: ED = 0 Expand up segment, offsets must be ≤ limit. ED = 1 Expand down segment, offsets must be > limit. W = 0 Data segment may not be written into. W = 1 Data segment may be written into.
2	Expansion Direction (ED)	
1	Writable (W)	
0	Accessed (A)	
3	Executable (E)	Code Segment Descriptor type is: Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged. R = 0 Code segment may not be read R = 1 Code segment may be read.
2	Conforming (C)	
1	Readable (R)	(S = 1, E = 1)
0	Accessed (A)	Segment has not been accessed. Segment selector has been loaded into segment register or used by selector test instructions.

Figure 11. Code and Data Segment Descriptor Formats

Code and data (including stack data) are stored in two types of segments: code segments and data segments. Both types are identified and defined by segment descriptors ($S = 1$). Code segments are identified by the executable (E) bit set to 1 in the descriptor access rights byte. The access rights byte of both code and data segment descriptor types have three fields in common: present (P) bit, Descriptor Privilege Level (DPL), and accessed (A) bit. If $P = 0$, any attempted use of this segment will cause a not-present exception. DPL specifies the privilege level of the segment descriptor. DPL controls when the descriptor may be used by a task (refer to privilege discussion below). The A bit shows whether the segment has been previously accessed for usage profiling, a necessity for virtual memory systems. The CPU will always set this bit when accessing the descriptor.

Data segments ($S = 1, E = 0$) may be either read-only or read-write as controlled by the W bit of the access rights byte. Read-only ($W = 0$) data segments may not be written into. Data segments may grow in two directions, as determined by the Expansion Direction (ED) bit: upwards ($ED = 0$) for data segments, and downwards ($ED = 1$) for a segment containing a stack. The limit field for a data segment descriptor is interpreted differently depending on the ED bit (see Figure 11).

A code segment ($S = 1, E = 1$) may be execute-only or execute/read as determined by the Readable (R) bit. Code segments may never be written into and execute-only code segments ($R = 0$) may not be read. A code segment may also have an attribute called conforming (C). A conforming code segment may be shared by programs that execute at different privilege levels. The DPL of a conforming code segment defines the range of privilege levels at which the segment may be executed (refer to privilege discussion below). The limit field identifies the last byte of a code segment.

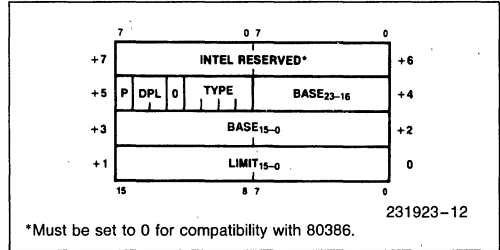
SYSTEM SEGMENT DESCRIPTORS ($S = 0, TYPE = 1-3$)

In addition to code and data segment descriptors, the protected mode 80C286 defines System Segment Descriptors. These descriptors define special system data segments which contain a table of descriptors (Local Descriptor Table Descriptor) or segments which contain the execution state of a task (Task State Segment Descriptor).

Figure 12 gives the formats for the special system data segment descriptors. The descriptors contain a 24-bit base address of the segment and a 16-bit limit. The access byte defines the type of descriptor, its state and privilege level. The descriptor contents are valid and the segment is in physical memory if $P = 1$. If $P = 0$, the segment is not valid. The DPL field is only used in Task State Segment descriptors and indicates the privilege level at which the descrip-

tor may be used (see Privilege). Since the Local Descriptor Table descriptor may only be used by a special privileged instruction, the DPL field is not used. Bit 4 of the access byte is 0 to indicate that it is a system control descriptor. The type field specifies the descriptor type as indicated in Figure 12.

System Segment Descriptor



System Segment Descriptor Fields

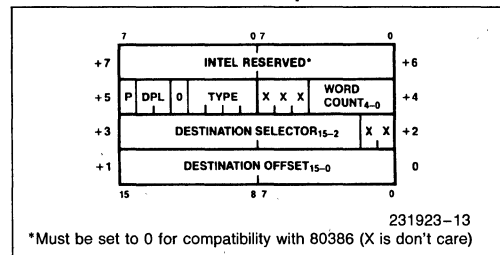
Name	Value	Description
TYPE	1	Available Task State Segment (TSS)
	2	Local Descriptor Table
	3	Busy Task State Segment (TSS)
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid
DPL	0-3	Descriptor Privilege Level
BASE	24-bit number	Base Address of special system data segment in real memory
LIMIT	16-bit number	Offset of last byte in segment

Figure 12. System Segment Descriptor Format

GATE DESCRIPTORS ($S = 0, TYPE = 4-7$)

Gates are used to control access to entry points within the target code segment. The gate descriptors are call gates, task gates, interrupt gates and trap gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the CPU to automatically perform protection checks and control entry point of the destination. Call gates are used to change privilege levels (see Privilege), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines. The interrupt gate disables interrupts (resets IF) while the trap gate does not.

Gate Descriptor



Gate Descriptor Fields

Name	Value	Description
TYPE	4	-Call Gate
	5	-Task Gate
	6	-Interrupt Gate
	7	-Trap Gate
P	0	-Descriptor Contents are not valid
	1	-Descriptor Contents are valid
DPL	0-3	Descriptor Privilege Level
WORD COUNT	0-31	Number of words to copy from callers stack to called procedures stack. Only used with call gate.
DESTINATION SELECTOR	16-bit selector	Selector to the target code segment (Call, Interrupt or Trap Gate) Selector to the target task state segment (Task Gate)
DESTINATION OFFSET	16-bit offset	Entry point within the target code segment

Figure 13. Gate Descriptor Format

Figure 13 shows the format of the gate descriptors. The descriptor contains a destination pointer that points to the descriptor of the target segment and the entry point offset. The destination selector in an interrupt gate, trap gate, and call gate must refer to a code segment descriptor. These gate descriptors contain the entry point to prevent a program from constructing and using an illegal entry point. Task gates may only refer to a task state segment. Since task gates invoke a task switch, the destination offset is not used in the task gate.

Exception 13 is generated when the gate is used if a destination selector does not refer to the correct descriptor type. The word count field is used in the call gate descriptor to indicate the number of parameters (0-31 words) to be automatically copied from the caller's stack to the stack of the called routine when a control transfer changes privilege levels. The word count field is not used by any other gate descriptor.

The access byte format is the same for all gate descriptors. P = 1 indicates that the gate contents are valid. P = 0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the de-

scriptor privilege level and specifies when this descriptor may be used by a task (refer to privilege discussion below). Bit 4 must equal 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 13.

SEGMENT DESCRIPTOR CACHE REGISTERS

A segment descriptor cache register is assigned to each of the four segment registers (CS, SS, DS, ES). Segment descriptors are automatically loaded (cached) into a segment descriptor cache register (Figure 14) whenever the associated segment register is loaded with a selector. Only segment descriptors may be loaded into segment descriptor cache registers. Once loaded, all references to that segment of memory use the cached descriptor information instead of reaccessing the descriptor. The descriptor cache registers are not visible to programs. No instructions exist to store their contents. They only change when a segment register is loaded.

SELECTOR FIELDS

A protected mode selector has three fields: descriptor entry index, local or global descriptor table indicator (TI), and selector privilege (RPL) as shown in Figure 15. These fields select one of two memory based tables of descriptors, select the appropriate table entry and allow highspeed testing of the selector's privilege attribute (refer to privilege discussion below).

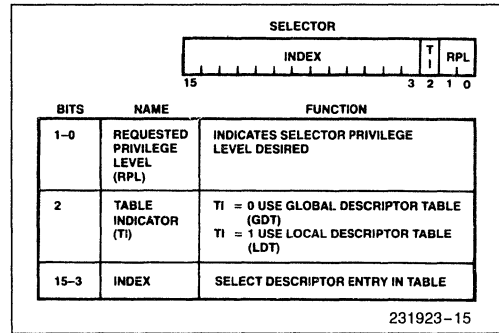


Figure 15. Selector Fields

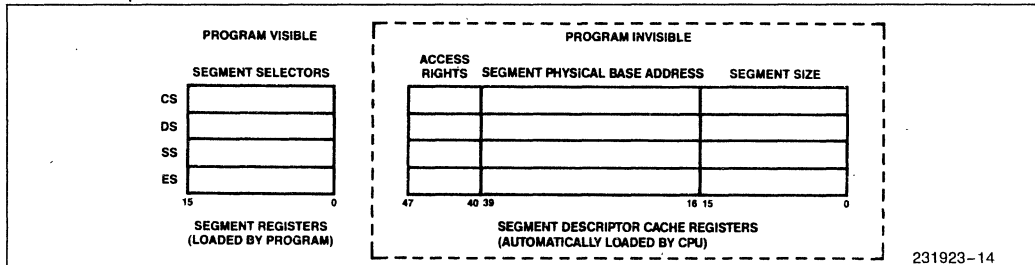


Figure 14. Descriptor Cache Registers

LOCAL AND GLOBAL DESCRIPTOR TABLES

Two tables of descriptors, called descriptor tables, contain all descriptors accessible by a task at any given time. A descriptor table is a linear array of up to 8192 descriptors. The upper 13 bits of the selector value are an index into a descriptor table. Each table has a 24-bit base register to locate the descriptor table in physical memory and a 16-bit limit register that confine descriptor access to the defined limits of the table as shown in Figure 16. A restartable exception (13) will occur if an attempt is made to reference a descriptor outside the table limits.

One table, called the Global Descriptor table (GDT), contains descriptors available to all tasks. The other table, called the Local Descriptor Table (LDT), contains descriptors that can be private to a task. Each task may have its own private LDT. The GDT may contain all descriptor types except interrupt and trap descriptors. The LDT may contain only segment, task gate, and call gate descriptors. A segment cannot be accessed by a task if its segment descriptor does not exist in either descriptor table at the time of access.

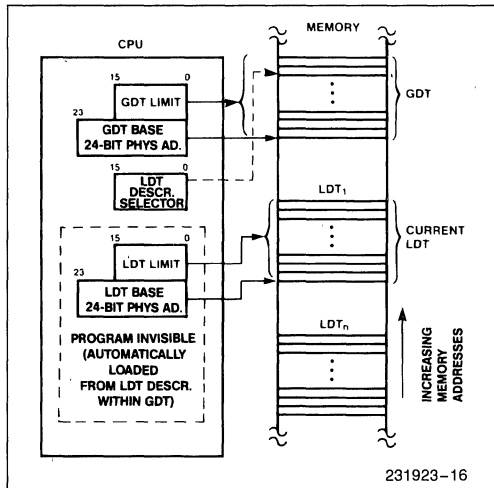


Figure 16. Local and Global Descriptor Table Definition

The LGDT and LLDT instructions load the base and limit of the global and local descriptor tables. LGDT and LLDT are privileged, i.e. they may only be executed by trusted programs operating at level 0. The LGDT instruction loads a six byte field containing the 16-bit table limit and 24-bit physical base address of the Global Descriptor Table as shown in Figure 17. The LDT instruction loads a selector which refers to a Local Descriptor Table descriptor containing the

base address and limit for an LDT, as shown in Figure 12.

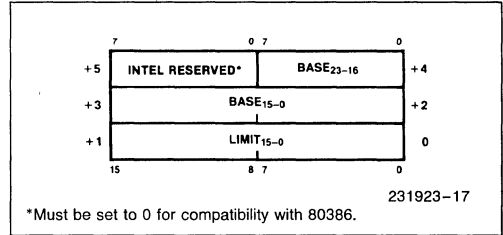


Figure 17. Global Descriptor Table and Interrupt Descriptor Table Data Type

INTERRUPT DESCRIPTOR TABLE

The protected mode 80C286 has a third descriptor table, called the Interrupt Descriptor Table (IDT) (see Figure 18), used to define up to 256 interrupts. It may contain only task gates, interrupt gates and trap gates. The IDT (Interrupt Descriptor Table) has a 24-bit physical base and 16-bit limit register in the CPU. The privileged LIDT instruction loads these registers with a six byte value of identical form to that of the LGDT instruction (see Figure 17 and Protected Mode Initialization).

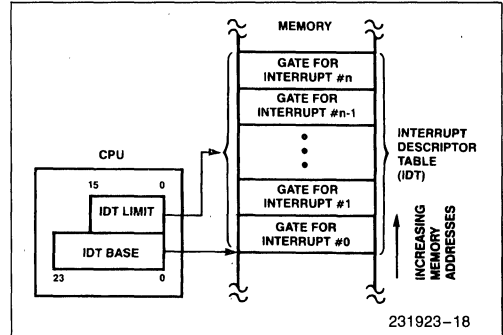
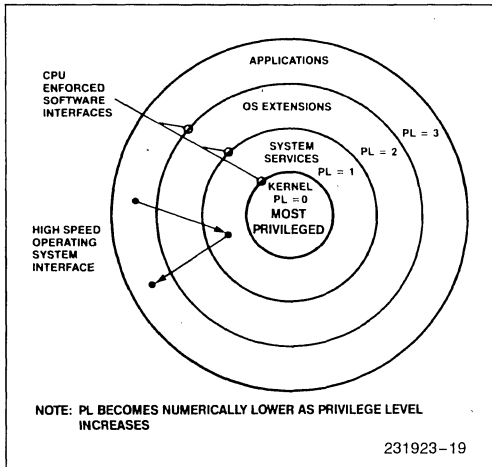


Figure 18. Interrupt Descriptor Table Definition

References to IDT entries are made via INT instructions, external interrupt vectors, or exceptions. The IDT must be at least 256 bytes in size to allocate space for all reserved interrupts.

Privilege

The 80C286 has a four-level hierarchical privilege system which controls the use of privileged instructions and access to descriptors (and their associated segments) within a task. Four-level privilege, as shown in Figure 19, is an extension of the user/supervisor mode commonly found in minicomputers. The privilege levels are numbered 0 through 3.



Level 0 is the most privileged level. Privilege levels provide protection within a task. (Tasks are isolated by providing private LDT's for each task.) Operating system routines, interrupt handlers, and other system software can be included and protected within the virtual address space of each task using the four levels of privilege. Each task in the system has a separate stack for each of its privilege levels.

Tasks, descriptors, and selectors have a privilege level attribute that determines whether the descriptor may be used. Task privilege effects the use of instructions and descriptors. Descriptor and selector privilege only effect access to the descriptor.

TASK PRIVILEGE

A task always executes at one of the four privilege levels. The task privilege level at any specific instant is called the Current Privilege Level (CPL) and is defined by the lower two bits of the CS register. CPL cannot change during execution in a single code segment. A task's CPL may only be changed by control transfers through gate descriptors to a new code segment (See Control Transfer). Tasks begin executing at the CPL value specified by the code segment selector within TSS when the task is initiated via a task switch operation (See Figure 20). A task executing at Level 0 can access all data segments defined in the GDT and the task's LDT and is considered the most trusted level. A task executing a Level 3 has the most restricted access to data and is considered the least trusted level.

DESCRIPTOR PRIVILEGE

Descriptor privilege is specified by the Descriptor Privilege Level (DPL) field of the descriptor access byte. DPL specifies the least trusted task privilege

level (CPL) at which a task may access the descriptor. Descriptors with DPL = 0 are the most protected. Only tasks executing at privilege level 0 (CPL = 0) may access them. Descriptors with DPL = 3 are the least protected (i.e. have the least restricted access) since tasks can access them when CPL = 0, 1, 2, or 3. This rule applies to all descriptors, except LDT descriptors.

SELECTOR PRIVILEGE

Selector privilege is specified by the Requested Privilege Level (RPL) field in the least significant two bits of a selector. Selector RPL may establish a less trusted privilege level than the current privilege level for the use of a selector. This level is called the task's effective privilege level (EPL). RPL can only reduce the scope of a task's access to data with this selector. A task's effective privilege is the numeric maximum of RPL and CPL. A selector with RPL = 0 imposes no additional restriction on its use while a selector with RPL = 3 can only refer to segments at privilege Level 3 regardless of the task's CPL. RPL is generally used to verify that pointer parameters passed to a more trusted procedure are not allowed to use data at a more privileged level than the caller (refer to pointer testing instructions).

Descriptor Access and Privilege Validation

Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL. The two basic types of segment accesses are control transfer (selectors loaded into CS) and data (selectors loaded into DS, ES or SS).

DATA SEGMENT ACCESS

Instructions that load selectors into DS and ES must refer to a data segment descriptor or readable code segment descriptor. The CPL of the task and the RPL of the selector must be the same as or more privileged (numerically equal to or lower than) than the descriptor DPL. In general, a task can only access data segments at the same or less privileged levels than the CPL or RPL (whichever is numerically higher) to prevent a program from accessing data it cannot be trusted to use.

An exception to the rule is a readable conforming code segment. This type of code segment can be read from any privilege level.

If the privilege checks fail (e.g. DPL is numerically less than the maximum of CPL and RPL) or an incorrect type of descriptor is referenced (e.g. gate de-

scriptor or execute only code segment) exception 13 occurs. If the segment is not present, exception 11 is generated.

Instructions that load selectors into SS must refer to data segment descriptors for writable data segments. The descriptor privilege (DPL) and RPL must equal CPL. All other descriptor types or a privilege level violation will cause exception 13. A not present fault causes exception 12.

CONTROL TRANSFER

Four types of control transfer can occur when a selector is loaded into CS by a control transfer operation (see Table 10). Each transfer type can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules (e.g. JMP through a call gate or RET to a Task State Segment) will cause exception 13.

The ability to reference a descriptor for control transfer is also subject to rules of privilege. A CALL or JUMP instruction may only reference a code segment descriptor with DPL equal to the task CPL or a conforming segment with DPL of equal or greater privilege than CPL. The RPL of the selector used to reference the code descriptor must have as much privilege as CPL.

RET and IRET instructions may only reference code segment descriptors with descriptor privilege equal to or less privileged than the task CPL. The selector loaded into CS is the return address from the stack. After the return, the selector RPL is the task's new CPL. If CPL changes, the old stack pointer is popped after the return address.

When a JMP or CALL references a Task State Segment descriptor, the descriptor DPL must be the same or less privileged than the task's CPL. Refer-

ence to a valid Task State Segment descriptor causes a task switch (see Task Switch Operation). Reference to a Task State Segment descriptor at a more privileged level than the task's CPL generates exception 13.

When an instruction or interrupt references a gate descriptor, the gate DPL must have the same or less privilege than the task CPL. If DPL is at a more privileged level than CPL, exception 13 occurs. If the destination selector contained in the gate references a code segment descriptor, the code segment descriptor DPL must be the same or more privileged than the task CPL. If not, Exception 13 is issued. After the control transfer, the code segment descriptors DPL is the task's new CPL. If the destination selector in the gate references a task state segment, a task switch is automatically performed (see Task Switch Operation).

The privilege rules on control transfer require:

- JMP or CALL direct to a code segment (code segment descriptor) can only be to a conforming segment with DPL of equal or greater privilege than CPL or a non-conforming segment at the same privilege level.
- interrupts within the task or calls that may change privilege levels, can only transfer control through a gate at the same or a less privileged level than CPL to a code segment at the same or more privileged level than CPL.
- return instructions that don't switch tasks can only return control to a code segment at the same or less privileged level.
- task switch can be performed by a call, jump or interrupt which references either a task gate or task state segment at the same or less privileged level.

Table 10. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL.	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag word) = 0
 **NT (Nested Task bit of flag word) = 1

PRIVILEGE LEVEL CHANGES

Any control transfer that changes CPL within the task, causes a change of stacks as part of the operation. Initial values of SS:SP for privilege levels 0, 1, and 2 are kept in the task state segment (refer to Task Switch Operation). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and SP registers and the previous stack pointer is pushed onto the new stack.

When returning to the original privilege level, its stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words, as specified in the gate, are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

Protection

The 80C286 includes mechanisms to protect critical instructions that affect the CPU execution state (e.g. HLT) and code or data segments from improper usage. These protection mechanisms are grouped into three forms:

Restricted *usage* of segments (e.g. no write allowed to read-only data segments). The only segments available for use are defined by descriptors in the Local Descriptor Table (LDT) and Global Descriptor Table (GDT).

Restricted *access* to segments via the rules of privilege and descriptor usage.

Privileged instructions or operations that may only be executed at certain privilege levels as determined by the CPL and I/O Privilege Level (IOPL). The IOPL is defined by bits 14 and 13 of the flag word.

These checks are performed for all instructions and can be split into three categories: segment load checks (Table 11), operand reference checks (Table 12), and privileged instruction checks (Table 13). Any violation of the rules shown will result in an exception. A not-present exception related to the stack segment causes exception 12.

The IRET and POPF instructions do not perform some of their defined functions if CPL is not of sufficient privilege (numerically small enough). Precisely these are:

- The IF bit is not changed if $CPL > IOPL$.
- The IOPL field of the flag word is not changed if $CPL > 0$.

No exceptions or other indication are given when these conditions occur.

Table 11. Segment Register Load Checks

Error Description	Exception Number
Descriptor table limit exceeded	13
Segment descriptor not-present	11 or 12
Privilege rules violated	13
Invalid descriptor/segment type segment register load: —Read only data segment load to SS —Special Control descriptor load to DS, ES, SS —Execute only segment load to DS, ES, SS —Data segment load to CS —Read/Execute code segment load to SS	13

Table 12. Operand Reference Checks

Error Description	Exception Number
Write into code segment	13
Read from execute-only code segment	13
Write to read-only data segment	13
Segment limit exceeded ¹	12 or 13

NOTE:
Carry out in offset calculations is ignored.

Table 13. Privileged Instruction Checks

Error Description	Exception Number
$CPL \neq 0$ when executing the following instructions: LIDT, LLDT, LGDT, LTR, LMSW, CTS, HLT	13
$CPL > IOPL$ when executing the following instructions: INS, IN, OUTS, OUT, STI, CLI, LOCK	13

EXCEPTIONS

The 80C286 detects several types of exceptions and interrupts, in protected mode (see Table 14). Most are restartable after the exceptional condition is removed. Interrupt handlers for most exceptions can read an error code, pushed on the stack after the return address, that identifies the selector involved (0 if none). The return address normally points to the failing instruction, including all leading prefixes. For a processor extension segment overrun exception, the return address will not point at the ESC instruction that caused the exception; however, the processor extension registers may contain the address of the failing instruction.

Table 14. Protected Mode Exceptions

Interrupt Vector	Function	Return Address At Falling Instruction?	Always Restartable?	Error Code on Stack?
8	Double exception detected	Yes	No ²	Yes
9	Processor extension segment overrun	No	No ²	No
10	Invalid task state segment	Yes	Yes	Yes
11	Segment not present	Yes	Yes	Yes
12	Stack segment overrun or stack segment not present	Yes	Yes ¹	Yes
13	General protection	Yes	No ²	Yes

NOTE:

1. When a PUSH or POP instruction attempts to wrap around the stack segment, the machine state after the exception will not be restartable because stack segment wrap around is not permitted. This condition is identified by the value of the saved SP being either 0000(H), 0001(H), FFFE(H), or FFFF(H).
2. These exceptions indicate a violation to privilege rules or usage rules has occurred. Restart is generally not attempted under those conditions.

These exceptions indicate a violation to privilege rules or usage rules has occurred. Restart is generally not attempted under those conditions.

All these checks are performed for all instructions and can be split into three categories: segment load checks (Table 11), operand reference checks (Table 12), and privileged instruction checks (Table 13). Any violation of the rules shown will result in an exception. A not-present exception causes exception 11 or 12 and is restartable.

Special Operations

TASK SWITCH OPERATION

The 80C286 provides a built-in task switch operation which saves the entire 80C286 execution state (registers, address space, and a link to the previous task), loads a new execution state, and commences execution in the new task. Like gates, the task switch operation is invoked by executing an intersegment JMP or CALL instruction which refers to a Task State Segment (TSS) or task gate descriptor in the GDT or LDT. An INT n instruction, exception, or external interrupt may also invoke the task switch operation by selecting a task gate descriptor in the associated IDT descriptor entry.

The TSS descriptor points at a segment (see Figure 20) containing the entire 80C286 execution state while a task gate descriptor contains a TSS selector. The limit field of the descriptor must be > 002B(H).

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 80C286 called the Task Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector.

The IRET instruction is used to return control to the task that called the current task or was interrupted. Bit 14 in the flag register is called the Nested Task (NT) bit. It controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular current task by popping values off the stack; when NT = 1, IRET performs a task switch operation back to the previous task.

When a CALL, JMP, or INT instruction initiates a task switch, the old (except for case of JMP) and new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. NT may also be set or cleared by POPF or IRET instructions.

The task state segment is marked busy by changing the descriptor type field from Type 1 to Type 3. Use of a selector that references a busy task state segment causes Exception 13.

PROCESSOR EXTENSION CONTEXT SWITCHING

The context of a processor extension (such as the 80287 numerics processor) is not changed by the task switch operation. A processor extension context need only be changed when a different task attempts to use the processor extension (which still contains the context of a previous task). The 80C286 detects the first use of a processor extension after a task switch by causing the processor extension not present exception (7). The interrupt handler may then decide whether a context change is necessary.

Whenever the 80C286 switches tasks, it sets the Task Switched (TS) bit of the MSW. TS indicates that a processor extension context may belong to a different task than the current one. The processor extension not present exception (7) will occur when attempting to execute an ESC or WAIT instruction if TS = 1 and a processor extension is present (MP = 1 in MSW).

POINTER TESTING INSTRUCTIONS

The 80C286 provides several instructions to speed pointer testing and consistency checks for maintaining system integrity (see Table 15). These instruc-

tions use the memory management hardware to verify that a selector value refers to an appropriate segment without risking an exception. A condition flag (ZF) indicates whether use of the selector or segment will cause an exception.

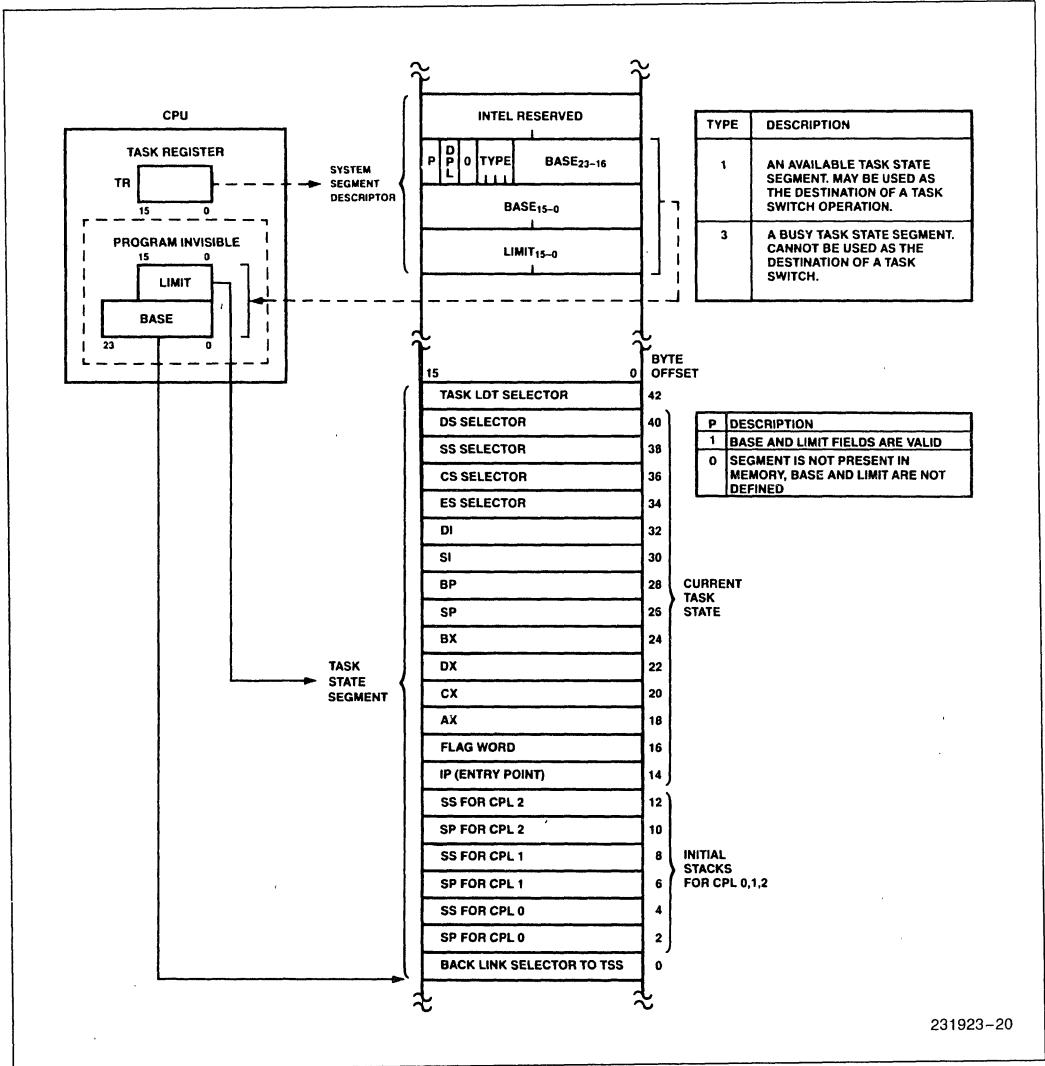


Figure 20. Task State Segment and TSS Registers

Table 15. 80C286 Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed by ARPL.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

DOUBLE FAULT AND SHUTDOWN

If two separate exceptions are detected during a single instruction execution, the 80C286 performs the double fault exception (8). If an execution occurs during processing of the double fault exception, the 80C286 will enter shutdown. During shutdown no further instructions or exceptions are processed. Either NMI (CPU remains in protected mode) or RESET (CPU exits protected mode) can force the 80C286 out of shutdown. Shutdown is externally signalled via a HALT bus operation with A₁ LOW.

PROTECTED MODE INITIALIZATION

The 80C286 initially executes in real address mode after RESET. To allow initialization code to be placed at the top of physical memory, A₂₃-A₂₀ will be HIGH when the 80C286 performs memory references relative to the CS register until CS is changed. A₂₃-A₂₀ will be zero for references to the DS, ES, or SS segments. Changing CS in real address mode will force A₂₃-A₂₀ LOW whenever CS is used again. The initial CS:IP value of F000:FFF0 provides 64K bytes of code space for initialization code without changing CS.

Protected mode operation requires several registers to be initialized. The GDT and IDT base registers must refer to a valid GDT and IDT. After executing the LMSW instruction to set PE, the 80C286 must

immediately execute an intra-segment JMP instruction to clear the instruction queue of instructions decoded in real address mode.

To force the 80C286 CPU registers to match the initial protected mode state assumed by software, execute a JMP instruction with a selector referring to the initial TSS used in the system. This will load the task register, local descriptor table register, segment registers and initial general register state. The TR should point at a valid TSS since any task switch operation involves saving the current task state.

SYSTEM INTERFACE

The 80C286 system interface appears in two forms: a local bus and a system bus. The local bus consists of address, data, status, and control signals at the pins of the CPU. A system bus is any buffered version of the local bus. A system bus may also differ from the local bus in terms of coding of status and control lines and/or timing and loading of signals. The 80C286 family includes several devices to generate standard system buses such as the IEEE 796 standard MULTIBUS.

Bus Interface Signals and Timing

The 80C286 microsystem local bus interfaces the 80C286 to local memory and I/O components. The interface has 24 address lines, 16 data lines, and 8 status and control signals.

The 80C286 CPU, 82C284 clock generator, 82C288 bus controller, transceivers, and latches provide a buffered and decoded system bus interface. The 82C284 generates the system clock and synchronizes $\overline{\text{READY}}$ and RESET. The 82C288 converts bus operation status encoded by the 80C286 into command and bus control signals. These components can provide the timing and electrical power drive levels required for most system bus interfaces including the Multibus.

Physical Memory and I/O Interface

A maximum of 16 megabytes of physical memory can be addressed in protected mode. One megabyte can be addressed in real address mode. Memory is accessible as bytes or words. Words consist of any two consecutive bytes addressed with the least significant byte stored in the lowest address.

Byte transfers occur on either half of the 16-bit local data bus. Even bytes are accessed over D₇-D₀ while odd bytes are transferred over D₁₅-D₈. Even-addressed words are transferred over D₁₅-D₀ in one bus cycle, while odd-addressed word require *two* bus operations. The first transfers data on D₁₅-D₈, and the second transfers data on D₇-D₀. Both byte data transfers occur automatically, transparent to software.

Two bus signals, A_0 and \overline{BHE} , control transfers over the lower and upper halves of the data bus. Even address byte transfers are indicated by A_0 LOW and \overline{BHE} HIGH. Odd address byte transfers are indicated by A_0 HIGH and \overline{BHE} LOW. Both A_0 and \overline{BHE} are LOW for even address word transfers.

The I/O address space contains 64K addresses in both modes. The I/O space is accessible as either bytes or words, as is memory. Byte wide peripheral devices may be attached to either the upper or lower byte of the data bus. Byte-wide I/O devices attached to the upper data byte ($D_{15}-D_8$) are accessed with odd I/O addresses. Devices on the lower data byte are accessed with even I/O addresses. An interrupt controller such as Intel's 82C59A-2 must be connected to the lower data byte (D_7-D_0) for proper return of the interrupt vector.

Bus Operation

The 80C286 uses a double frequency system clock (CLK input) to control bus timing. All signals on the local bus are measured relative to the system CLK input. The CPU divides the system clock by 2 to produce the internal processor clock, which determines bus state. Each processor clock is composed of two system clock cycles named phase 1 and phase 2. The 82C284 clock generator output (PCLK) identifies the next phase of the processor clock. (See Figure 21.)

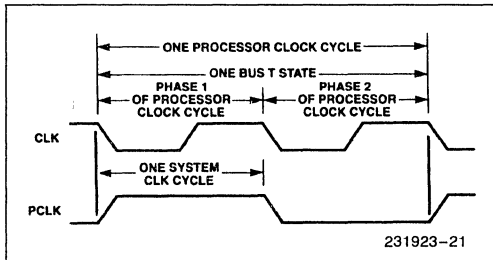


Figure 21. System and Processor Clock Relationships

Six types of bus operations are supported; memory read, memory write, I/O read, I/O write, interrupt acknowledgement, and halt/shutdown. Data can be transferred at a maximum rate of one word per two processor clock cycles.

The 80C286 bus has three basic states: idle (T_i), send status (T_s), and perform command (T_c). The 80C286 CPU also has a fourth local bus state called hold (T_h). T_h indicates that the 80C286 has surrendered control of the local bus to another bus master in response to a HOLD request.

Each bus state is one processor clock long. Figure 22 shows the four 80C286 local bus states and allowed transitions.

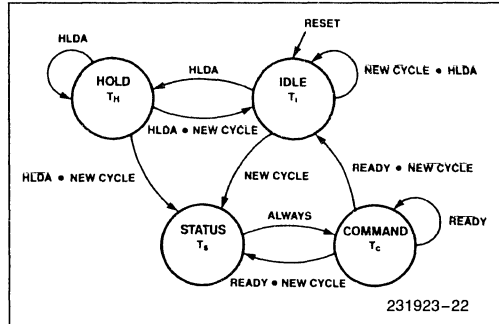


Figure 22. 80C286 Bus States

Bus States

The idle (T_i) state indicates that no data transfers are in progress or requested. The first active state T_s is signaled by status line $\overline{S1}$ or $\overline{S0}$ going LOW and identifying phase 1 of the processor clock. During T_s , the command encoding, the address, and data (for a write operation) are available on the 80C286 output pins. The 82C288 bus controller decodes the status signals and generates Multibus compatible read/write command and local transceiver control signals.

After T_s , the perform command (T_c) state is entered. Memory or I/O devices respond to the bus operation during T_c , either transferring read data to the CPU or accepting write data. T_c states may be repeated as often as necessary to assure sufficient time for the memory or I/O device to respond. The \overline{READY} signal determines whether T_c is repeated. A repeated T_c state is called a wait state.

During hold (T_h), the 80C286 will float* all address, data, and status output pins enabling another bus master to use the local bus. The 80C286 HOLD input signal is used to place the 80C286 into the T_h state. The 80C286 HLDA output signal indicates that the CPU has entered T_h .

Pipelined Addressing

The 80C286 uses a local bus interface with pipelined timing to allow as much time as possible for data access. Pipelined timing allows a new bus operation to be initiated every two processor cycles, while allowing each individual bus operation to last for three processor cycles.

The timing of the address outputs is pipelined such that the address of the next bus operation becomes available during the current bus operation. Or in other words, the first clock of the next bus operation is overlapped with the last clock of the current bus operation. Therefore, address decode and routing logic can operate in advance of the next bus operation.

*NOTE: See section on bus hold circuitry.

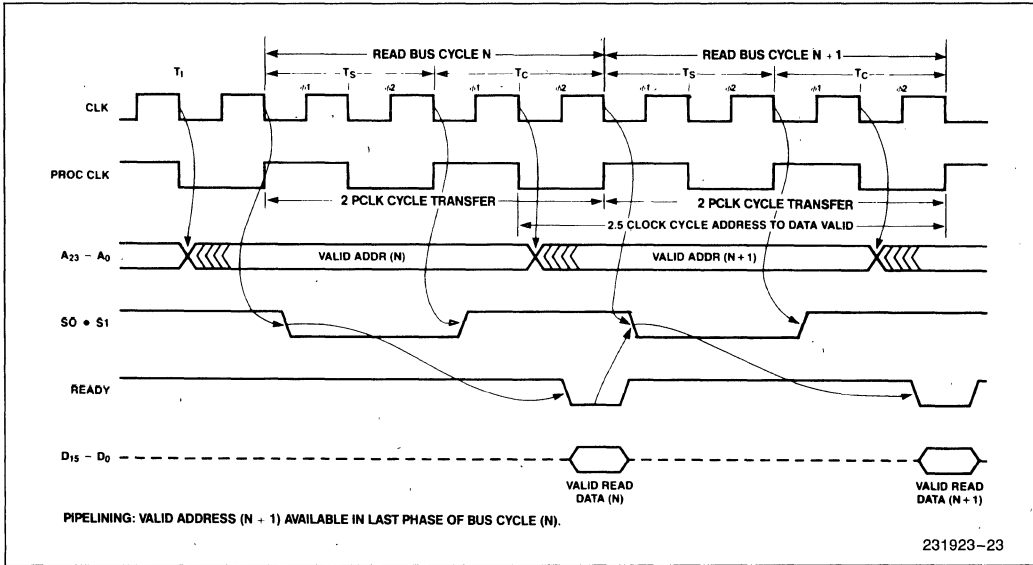


Figure 23. Basic Bus Cycle

External address latches may hold the address stable for the entire bus operation, and provide additional AC and DC buffering.

The 80C286 does not maintain the address of the current bus operation during all T_C states. Instead, the address for the next bus operation may be emitted during phase 2 of any T_C . The address remains valid during phase 1 of the first T_C to guarantee hold time, relative to ALE, for the address latch inputs.

Bus Control Signals

The 82C288 bus controller provides control signals; address latch enable (ALE), Read/Write commands, data transmit/receive (DT/R), and data enable (DEN) that control the address latches, data transceivers, write enable, and output enable for memory and I/O systems.

The Address Latch Enable (ALE) output determines when the address may be latched. ALE provides at least one system CLK period of address hold time from the end of the previous bus operation until the address for the next bus operation appears at the latch outputs. This address hold time is required to support MULTIBUS and common memory systems.

The data bus transceivers are controlled by 82C288 outputs Data Enable (DEN) and Data Transmit/Receive (DT/R). DEN enables the data transceivers; while DT/R controls tranceiver direction. DEN and DT/R are timed to prevent bus contention between the bus master, data bus transceivers, and system data bus transceivers.

Command Timing Controls

Two system timing customization options, command extension and command delay, are provided on the 80C286 local bus.

Command extension allows additional time for external devices to respond to a command and is analogous to inserting wait states on the 8086. External logic can control the duration of any bus operation such that the operation is only as long as necessary. The READY input signal can extend any bus operation for as long as necessary.

Command delay allows an increase of address or write data setup time to system bus command active for any bus operation by delaying when the system bus command becomes active. Command delay is controlled by the 82C288 CMDLY input. After T_S , the bus controller samples CMDLY at each falling edge of CLK. If CMDLY is HIGH, the 82C288 will not activate the command signal. When CMDLY is LOW, the 82C288 will activate the command signal. After the command becomes active, the CMDLY input is not sampled.

When a command is delayed, the available response time from command active to return read data or accept write data is less. To customize system bus timing, an address decoder can determine which bus operations require delaying the command. The CMDLY input does not affect the timing of ALE, DEN, or DT/R.

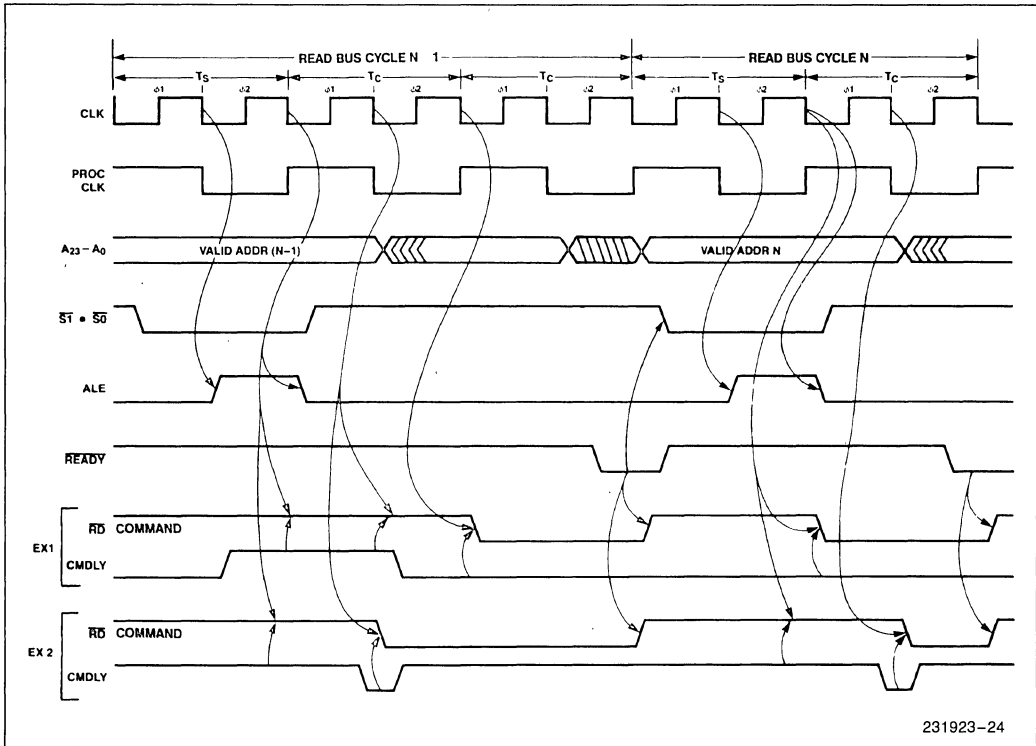


Figure 24. CMDLY Controls the Leading Edge of Command Signal

Figure 24 illustrates four uses of CMDLY. Example 1 shows delaying the read command two system CLKs for cycle N-1 and no delay for cycle N, and example 2 shows delaying the read command one system CLK for cycle N-1 and one system CLK delay for cycle N.

Bus Cycle Termination

At maximum transfer rates, the 80C286 bus alternates between the status and command states. The bus status signals become inactive after T_S so that they may correctly signal the start of the next bus operation after the completion of the current cycle. No external indication of T_C exists on the 80C286 local bus. The bus master and bus controller enter T_C directly after T_S and continue executing T_C cycles until terminated by $\overline{\text{READY}}$.

READY Operation

The current bus master and 82C288 bus controller terminate each bus operation simultaneously to achieve maximum bus operation bandwidth. Both are informed in advance by $\overline{\text{READY}}$ active (open-collector output from 82C284) which identifies the last T_C cycle of the current bus operation. The bus master and bus controller must see the same sense

of the $\overline{\text{READY}}$ signal, thereby requiring $\overline{\text{READY}}$ be synchronous to the system clock.

Synchronous Ready

The 82C284 clock generator provides $\overline{\text{READY}}$ synchronization from both synchronous and asynchronous sources (see Figure 25). The synchronous ready input ($\overline{\text{SRDY}}$) of the clock generator is sampled with the falling edge of CLK at the end of phase 1 of each T_C . The state of $\overline{\text{SRDY}}$ is then broadcast to the bus master and bus controller via the $\overline{\text{READY}}$ output line.

Asynchronous Ready

Many systems have devices or subsystems that are asynchronous to the system clock. As a result, their ready outputs cannot be guaranteed to meet the 82C284 $\overline{\text{SRDY}}$ setup and hold time requirements. But the 82C284 asynchronous ready input ($\overline{\text{ARDY}}$) is designed to accept such signals. The $\overline{\text{ARDY}}$ input is sampled at the beginning of each T_C cycle by 82C284 synchronization logic. This provides one system CLK cycle time to resolve its value before broadcasting it to the bus master and bus controller.

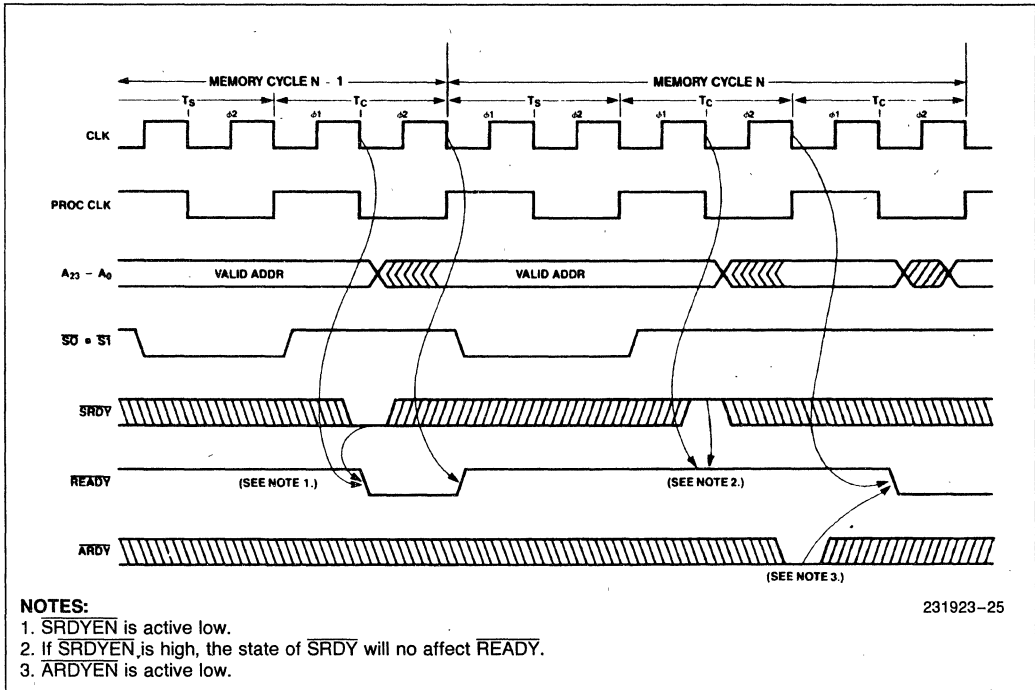


Figure 25. Synchronous and Asynchronous Ready

\overline{ARDY} or \overline{ARDYEN} must be HIGH at the end of T_s . \overline{ARDY} cannot be used to terminate bus cycle with no wait states.

Each ready input of the 82C284 has an enable pin (\overline{SRDYEN} and \overline{ARDYEN}) to select whether the current bus operation will be terminated by the synchronous or asynchronous ready. Either of the ready inputs may terminate a bus operation. These enable inputs are active low and have the same timing as their respective ready inputs. Address decode logic usually selects whether the current bus operation should be terminated by \overline{ARDY} or \overline{SRDY} .

Data Bus Control

Figures 26, 27, and 28 show how the DT/\overline{R} , DEN, data bus, and address signals operate for different combinations of read, write, and idle bus operations. DT/\overline{R} goes active (LOW) for a read operation. DT/\overline{R} remains HIGH before, during, and between write operations.

The data bus is driven with write data during the second phase of T_s . The delay in write data timing allows the read data drivers, from a previous read cycle, sufficient time to enter 3-state OFF* before the 80C286 CPU begins driving the local data bus for write operations. Write data will always remain valid for one system clock past the last T_c to provide sufficient hold time for Multibus or other similar memory or I/O systems. During write-read or write-idle sequences the data bus enters 3-state OFF* during the second phase of the processor cycle after the last T_c . In a write-write sequence the data bus does not enter 3-state OFF* between T_c and T_s .

Bus Usage

The 80C286 local bus may be used for several functions: instruction data transfers, data transfers by other bus masters, instruction fetching, processor extension data transfers, interrupt acknowledge, and halt/shutdown. This section describes local bus activities which have special signals or requirements.

*NOTE: See section on bus hold circuitry.

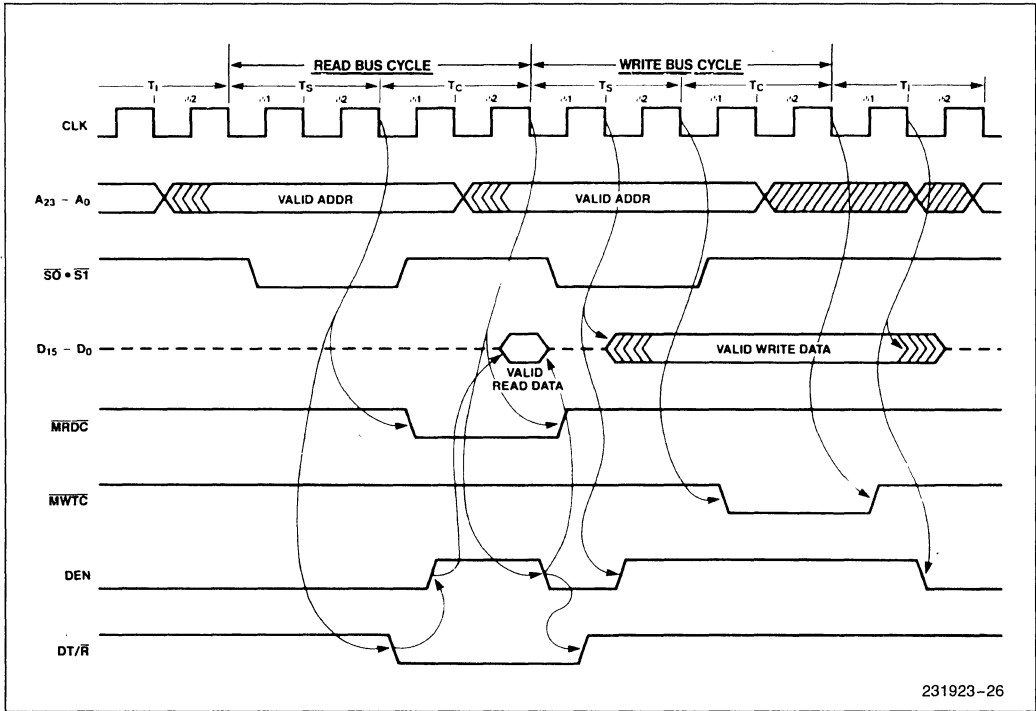


Figure 26. Back to Back Read-Write Cycles

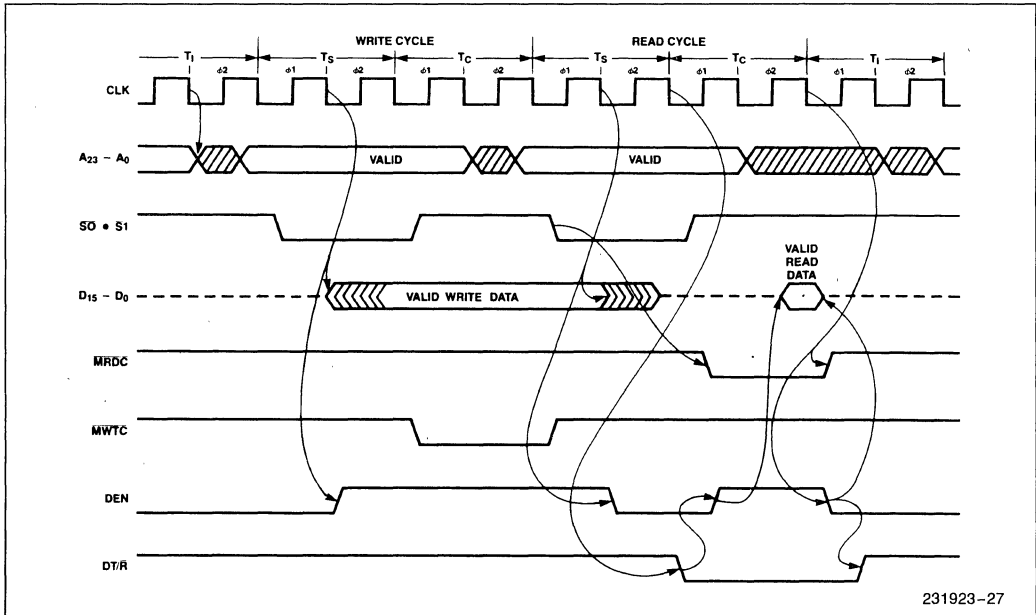


Figure 27. Back to Back Write-Read Cycles

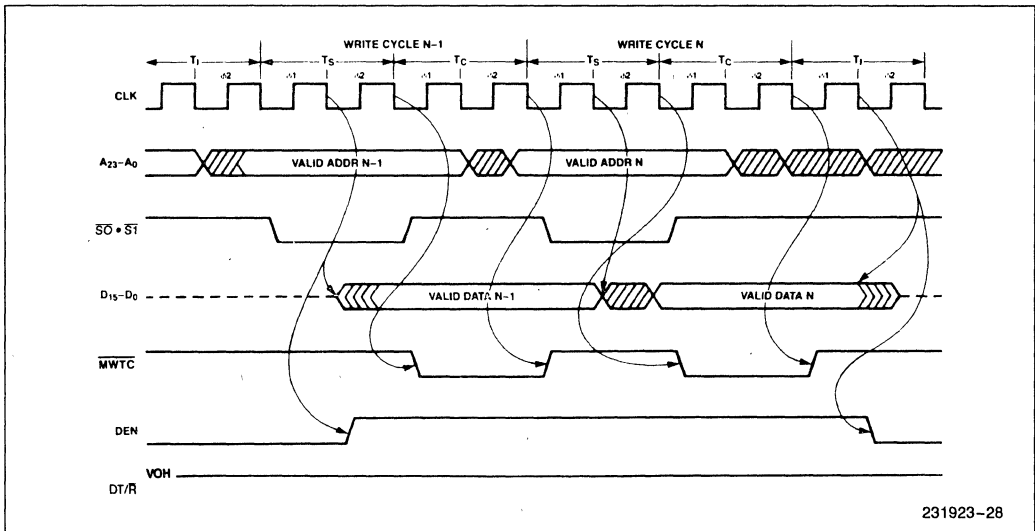


Figure 28. Back to Back Write-Write Cycles

231923-28

HOLD and HLDA

HOLD AND HLDA allow another bus master to gain control of the local bus by placing the 80C286 bus into the T_H state. The sequence of events required to pass control between the 80C286 and another local bus master are shown in Figure 29.

In this example, the 80C286 is initially in the T_H state as signaled by HLDA being active. Upon leaving T_H , as signaled by HLDA going inactive, a write operation is started. During the write operation another local bus master requests the local bus from the 80C286 as shown by the HOLD signal. After completing the write operation, the 80C286 performs one T_1 bus cycle, to guarantee write data hold time, then enters T_H as signaled by HLDA going active.

The \overline{CMDLY} signal and \overline{ARDY} ready are used to start and stop the write bus command, respectively. Note that \overline{SRDY} must be inactive or disabled by \overline{SRDYEN} to guarantee \overline{ARDY} will terminate the cycle.

HOLD must not be active during the time from the leading edge of RESET until 34 CLKs following the trailing edge of RESET.

Lock

The CPU asserts an active lock signal during Interrupt-Acknowledge cycles, the XCHG instruction, and during some descriptor accesses. Lock is also asserted when the LOCK prefix is used. The LOCK prefix may be used with the following ASM-286 assembly instructions; MOVS, INS, and OUTS. For bus cycles other than Interrupt-Acknowledge cycles,

Lock will be active for the first and subsequent cycles of a series of cycles to be locked. Lock will not be shown active during the last cycle to be locked. For the next-to-last cycle, Lock will become inactive at the end of the first T_C regardless of the number of wait-states inserted. For Interrupt-Acknowledge cycles, Lock will be active for each cycle, and will become inactive at the end of the first T_C for each cycle regardless of the number of wait-states inserted.

Instruction Fetching

The 80C286 Bus Unit (BU) will fetch instructions ahead of the current instruction being executed. This activity is called prefetching. It occurs when the local bus would otherwise be idle and obeys the following rules:

A prefetch bus operation starts when at least two bytes of the 6-byte prefetch queue are empty.

The prefetcher normally performs word prefetches independent of the byte alignment of the code segment base in physical memory.

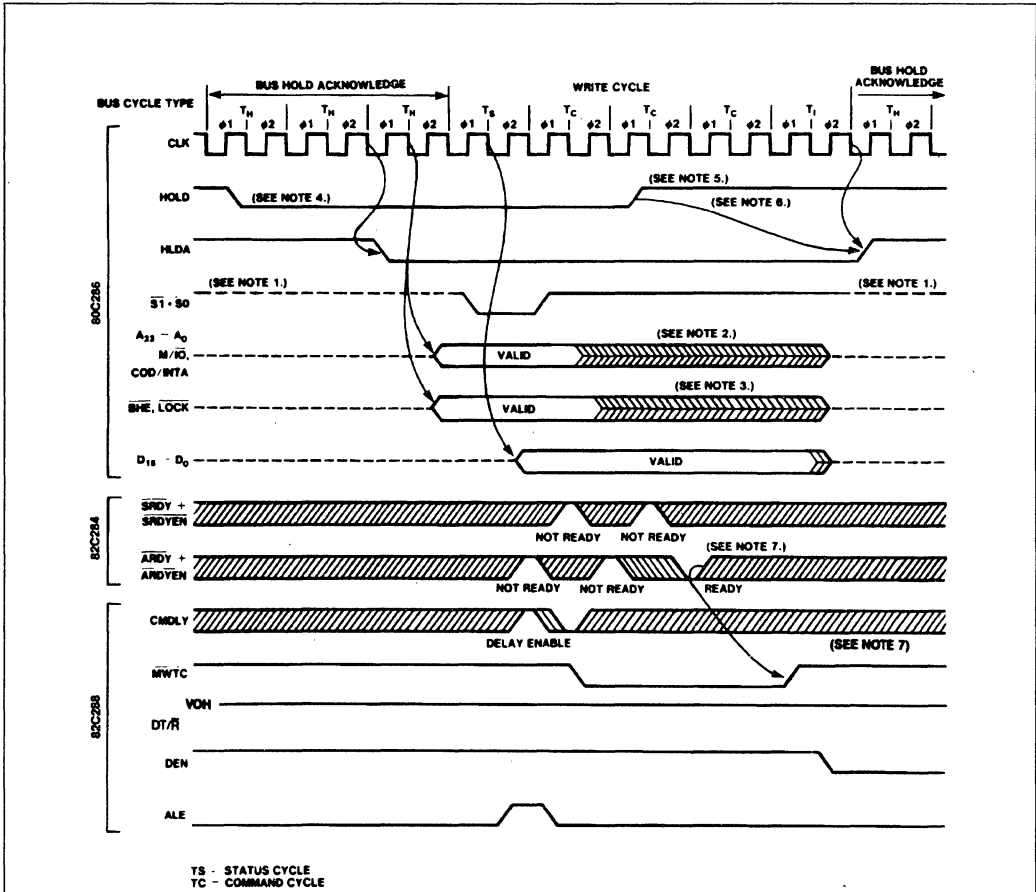
The prefetcher will perform only a byte code fetch operation for control transfers to an instruction beginning on a numerically odd physical address.

Prefetching stops whenever a control transfer or HLT instruction is decoded by the IU and placed into the instruction queue.

In real address mode, the prefetcher may fetch up to 6 bytes beyond the last control transfer or HLT instruction in a code segment.

In protected mode, the prefetcher will never cause a segment overrun exception. The prefetcher stops at the last physical memory word of the code segment. Exception 13 will occur if the program attempts to execute beyond the last full instruction in the code segment.

If the last byte of a code segment appears on an even physical memory address, the prefetcher will read the next physical byte of memory (perform a word code fetch). The value of this byte is ignored and any attempt to execute it causes exception 13.



TS - STATUS CYCLE
TC - COMMAND CYCLE

231923-29

NOTES:

1. Status lines are not driven by 80C286, yet remain high due to internal pullup resistors during HOLD state. See section on bus hold circuitry.
2. Address, M/IO and COD/INTA may start floating during any TC depending on when internal 80C286 bus arbiter decides to release bus to external HOLD. The float starts in φ2 of TC. See section on bus hold circuitry.
3. BHE and LOCK may start floating after the end of any TC depending on when internal 80C286 bus arbiter decides to release bus to external HOLD. The float starts in φ1 of TC. See section on bus hold circuitry.
4. The minimum HOLD to HLDA time is shown. Maximum is one TH longer.
5. The earliest HOLD time is shown. It will always allow a subsequent memory cycle if pending is shown.
6. The minimum HOLD to HLDA time is shown. Maximum is a function of the instruction, type of bus cycle and other machine state (i.e., Interrupts, Waits, Lock, etc.).
7. Asynchronous ready allows termination of the cycle. Synchronous ready does not signal ready in this example. Synchronous ready state is ignored after ready is signaled via the asynchronous input.

Figure 29. MULTIBUS® Write Terminated by Asynchronous Ready with Bus Hold

Processor Extension Transfers

The processor extension interface uses I/O port addresses 00F8(H), 00FA(H), and 00FC(H) which are part of the I/O port address range reserved by Intel. An ESC instruction with Machine Status Word bits EM = 0 and TS = 0 will perform I/O bus operations to one or more of these I/O port addresses independent of the value of IOPL and CPL.

ESC instructions with memory references enable the CPU to accept PEREQ inputs for processor extension operand transfers. The CPU will determine the operand starting address and read/write status of the instruction. For each operand transfer, two or three bus operations are performed, one word transfer with I/O port address 00FA(H) and one or two bus operations with memory. Three bus operations are required for each word operand aligned on an odd byte address.

NOTE:

Odd-aligned numeric instructions should be avoided when using an 80C286 system running six or more memory-write wait-states. The 80C286 can generate an incorrect numerics address if all the following conditions are met:

- Two floating point (FP) instructions are fetched and in the 80C286 queue.
- The first FP instruction is any floating point store except FSTSW AX.
- The second FP instruction is any floating point store except FSTSW AX.
- The second FP instruction accesses memory.
- The operand of the first instruction is aligned on an odd memory address.
- More than five wait-states are inserted during either of the last two memory write transfers (transferred as two bytes for odd aligned operands) of the first instruction.

The second FP instruction operand address will be incremented by one if these conditions are met. These conditions are most likely to occur in a multi-master system. For a hardware solution, contact your local Intel representative.

Ten or more command delays should not be used when accessing the numerics coprocessor. Excessive command delays can cause the 80C286 and 80287 to lose synchronization.

Interrupt Acknowledge Sequence

Figure 30 illustrates an interrupt acknowledge sequence performed by the 80C286 in response to an

INTR input. An interrupt acknowledge sequence consists of two INTA bus operations. The first allows a master 82C59A-2 Programmable Interrupt Controller (PIC) to determine which if any of its slaves should return the interrupt vector. An eight bit vector is read on D0–D7 of the 80C286 during the second INTA bus operation to select an interrupt handler routine from the interrupt table.

The Master Cascade Enable (MCE) signal of the 82C288 is used to enable the cascade address drivers, during INTA bus operations (See Figure 30), onto the local address bus for distribution to slave interrupt controllers via the system address bus. The 80C286 emits the $\overline{\text{LOCK}}$ signal (active LOW) during T_s of the first INTA bus operation. A local bus “hold” request will not be honored until the end of the second INTA bus operation.

Three idle processor clocks are provided by the 80C286 between INTA bus operations to allow for the minimum INTA to INTA time and CAS (cascade address) out delay of the 82C59A-2. The second INTA bus operation must always have at least one extra T_c state added via logic controlling $\overline{\text{READY}}$. This is needed to meet the 82C59A-2 minimum INTA pulse width.

Local Bus Usage Priorities

The 80C286 local bus is shared among several internal units and external HOLD requests. In case of simultaneous requests, their relative priorities are:

(Highest) Any transfers which assert $\overline{\text{LOCK}}$ either explicitly (via the LOCK instruction prefix) or implicitly (i.e. some segment descriptor accesses, interrupt acknowledge sequence, or an XCHG with memory).

The second of the two byte bus operations required for an odd aligned word operand.

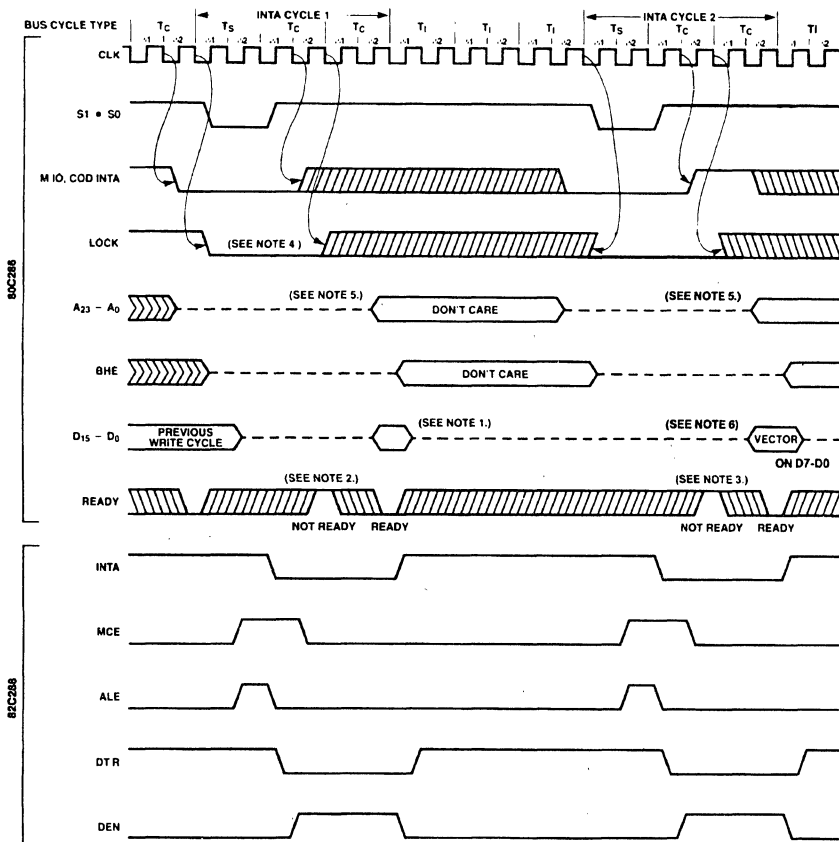
The second or third cycle of a processor extension data transfer.

Local bus request via HOLD input.

Processor extension data operand transfer via PEREQ input.

Data transfer performed by EU as part of an instruction.

(Lowest) An instruction prefetch request from BU. The EU will inhibit prefetching two processor clocks in advance of any data transfers to minimize waiting by EU for a prefetch to finish.



231923-32

NOTES:

1. Data is ignored, upper data bus, D₈-D₁₅, should not change state during this time.
2. First INTA cycle should have at least one wait state inserted to meet 8259A minimum INTA pulse width.
3. Second INTA cycle should have at least one wait state inserted to meet 8259A minimum INTA pulse width.
4. LOCK is active for the first INTA cycle to prevent a bus arbiter from releasing the bus between INTA cycles in a multi-master system. LOCK is also active for the second INTA cycle.
5. A₂₃-A₀ exits 3-state OFF during φ₂ of the second T_c in the INTA cycle. See section on bus hold circuitry.
6. Upper data bus should not change state during this time.

Figure 30. Interrupt Acknowledge Sequence

Halt or Shutdown Cycles

The 80C286 externally indicates halt or shutdown conditions as a bus operation. These conditions occur due to a HLT instruction or multiple protection exceptions while attempting to execute one instruction. A halt or shutdown bus operation is signalled when S₁, S₀ and COD/INTA are LOW and M/IO is HIGH. A₁ HIGH indicates halt, and A₁ LOW indicates shutdown. The 82C288 bus controller does not issue ALE, nor is READY required to terminate a halt or shutdown bus operation.

During halt or shutdown, the 80C286 may service PEREQ or HOLD requests. A processor extension segment overrun exception during shutdown will inhibit further service of PEREQ. Either NMI or RESET will force the 80C286 out of either halt or shutdown. An INTR, if interrupts are enabled, or a processor extension segment overrun exception will also force the 80C286 out of halt.

THE POWER-DOWN FEATURE OF THE 80C286

The 80C286, unlike the HMOS part, can enter into a power-down mode. By stopping the processor CLK, the processor will enter a power-down mode. Once in the power-down mode, all 80C286 outputs remain static (the same state as before the mode was entered). The 80C286 D.C. specification I_{CCS} rates the amount of current drawn by the processor when in the power-down mode. When the CLK is reapplied to the processor, it will resume execution where it was interrupted.

In order to obtain maximum benefits from the power-down mode, certain precautions should be taken. When in the power-down mode, all 80C286 outputs remain static and any output that is turned on and remains in a HIGH condition will source current when loaded. Best low-power performance can be obtained by first putting the processor in the HOLD

condition (turning off all of the output buffers), and then stopping the processor CLK in the phase 2 state. In this condition, any output that is loaded will source only the "Bus Hold Sustaining Current".

When stopping the processor clock, minimum clock high and low times cannot be violated (no glitches on the clock line).

Violating this condition can cause the 80C286 to erase its internal register states. Note that all inputs to the 80C286 (CLK, HOLD, PEREQ, RESET, READY, INTR, NMI, BUSY, and ERROR) should be at V_{CC} or V_{SS} ; any other value will cause the 80C286 to draw additional current.

When coming out of power-down mode, the system CLK must be started with the same polarity in which it was stopped. An example power down sequence is shown in Figure 31.

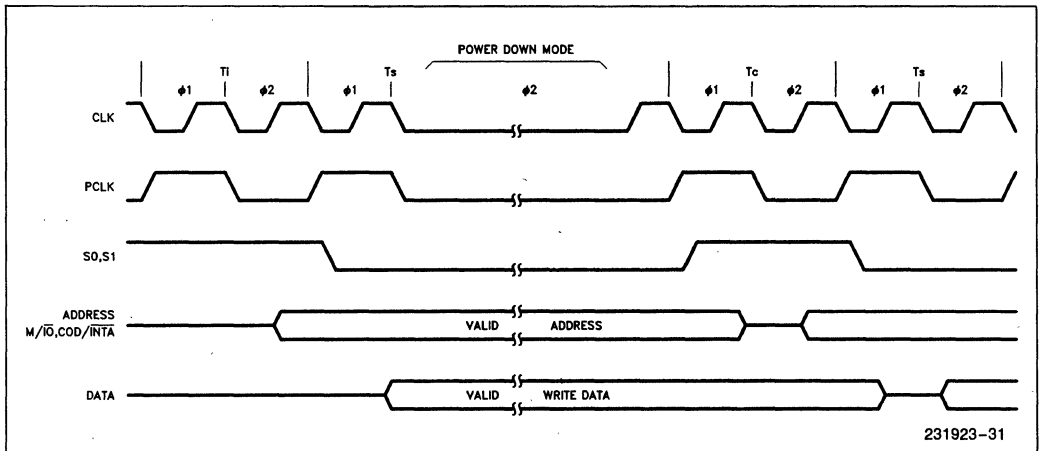


Figure 31. Example Power-Down Sequence

BUS HOLD CIRCUITRY

To avoid high current conditions caused by floating inputs to peripheral CMOS devices and eliminate the need for pull-up/down resistors, "bus-hold" circuitry has been used on all tri-state 80C286 outputs. See Table A for a list of these pins and Figures Ba and Bb for a complete description of which pins have bus hold circuitry. These circuits will maintain the last valid logic state if no driving source is present (i.e., an unconnected pin or a driving source which goes to a high impedance state). To overdrive the "bus hold" circuits, an external driver must be capable of supplying the maximum "Bus Hold Overdrive" sink or source current at valid input voltage levels. Since this "bus hold" circuitry is active and not a

"resistive" type element, the associated power supply current is negligible and power dissipation is significantly reduced when compared to the use of passive pull-up resistors.

Bus Hold Circuitry on the 80C286

Signal	Pin Location	Polarity Pulled to when tri-stated
S1, S0, PEACK, LOCK	4-6, 68	Hi, See Figure Bb
Data Bus (D ₀ -D ₁₅)	36-51	Hi/Lo, See Figure Ba
COD/INTA, M/I \bar{O}	66-67	Hi/Lo, See Figure Ba

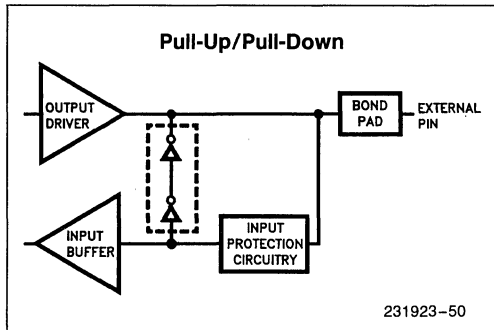


Figure Ba. Bus Hold Circuitry Pins 36-51, 66-67

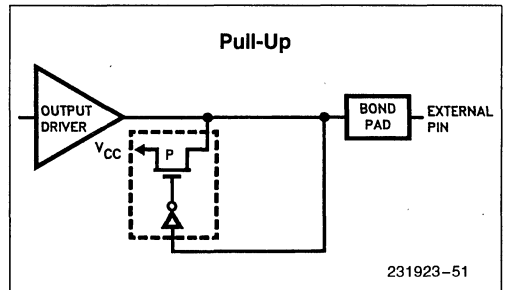


Figure Bb. Bus Hold Circuitry Pins 4-6, 68

SYSTEM CONFIGURATIONS

The versatile bus structure of the 80C286 microsystem, with a full complement of support chips, allows flexible configuration of a wide range of systems. The basic configuration, shown in Figure 32, is similar to an 8086 maximum mode system. It includes the CPU plus an 82C59A-2 interrupt controller, 82C284 clock generator, and the 82C288 Bus Controller.

As indicated by the dashed lines in Figure 32, the ability to add processor extensions is an integral feature of 80C286 microsystems. The processor extension interface allows external hardware to perform special functions and transfer data concurrent with CPU execution of other instructions. Full system integrity is maintained because the 80C286 supervises all data transfers and instruction execution for the processor extension.

The 80287 has all the instructions and data types of an 8087. The 80287 NPX can perform numeric calculations and data transfers concurrently with CPU program execution. Numerics code and data have the same integrity as all other information protected by the 80C286 protection mechanism.

The 80C286 can overlap chip select decoding and address propagation during the data transfer for the previous bus operation. This information is latched by ALE during the middle of a T_s cycle. The latched chip select and address information remains stable during the bus operation while the next cycle's ad-

dress is being decoded and propagated into the system. Decode logic can be implemented with a high speed PROM or PAL.

The optional decode logic shown in Figure 32 takes advantage of the overlap between address and data of the 80C286 bus cycle to generate advanced memory and IO-select signals. This minimizes system performance degradation caused by address propagation and decode delays. In addition to selecting memory and I/O, the advanced selects may be used with configurations supporting local and system buses to enable the appropriate bus interface for each bus cycle. The $\overline{COD}/\overline{INTA}$ and M/\overline{IO} signals are applied to the decode logic to distinguish between interrupt, I/O, code and data bus cycles.

By adding a bus arbiter, the 80C286 provides a MULTIBUS system bus interface as shown in Figure 33. The ALE output of the 82C288 for the MULTIBUS bus is connected to its CMDLY input to delay the start of commands one system CLK as required to meet MULTIBUS address and write data setup times. This arrangement will add at least one extra T_c state to each bus operation which uses the MULTIBUS.

A second 82C288 bus controller and additional latches and transceivers could be added to the local bus of Figure 33. This configuration allows the 80C286 to support an on-board bus for local memory and peripherals, and the MULTIBUS for system bus interfacing.

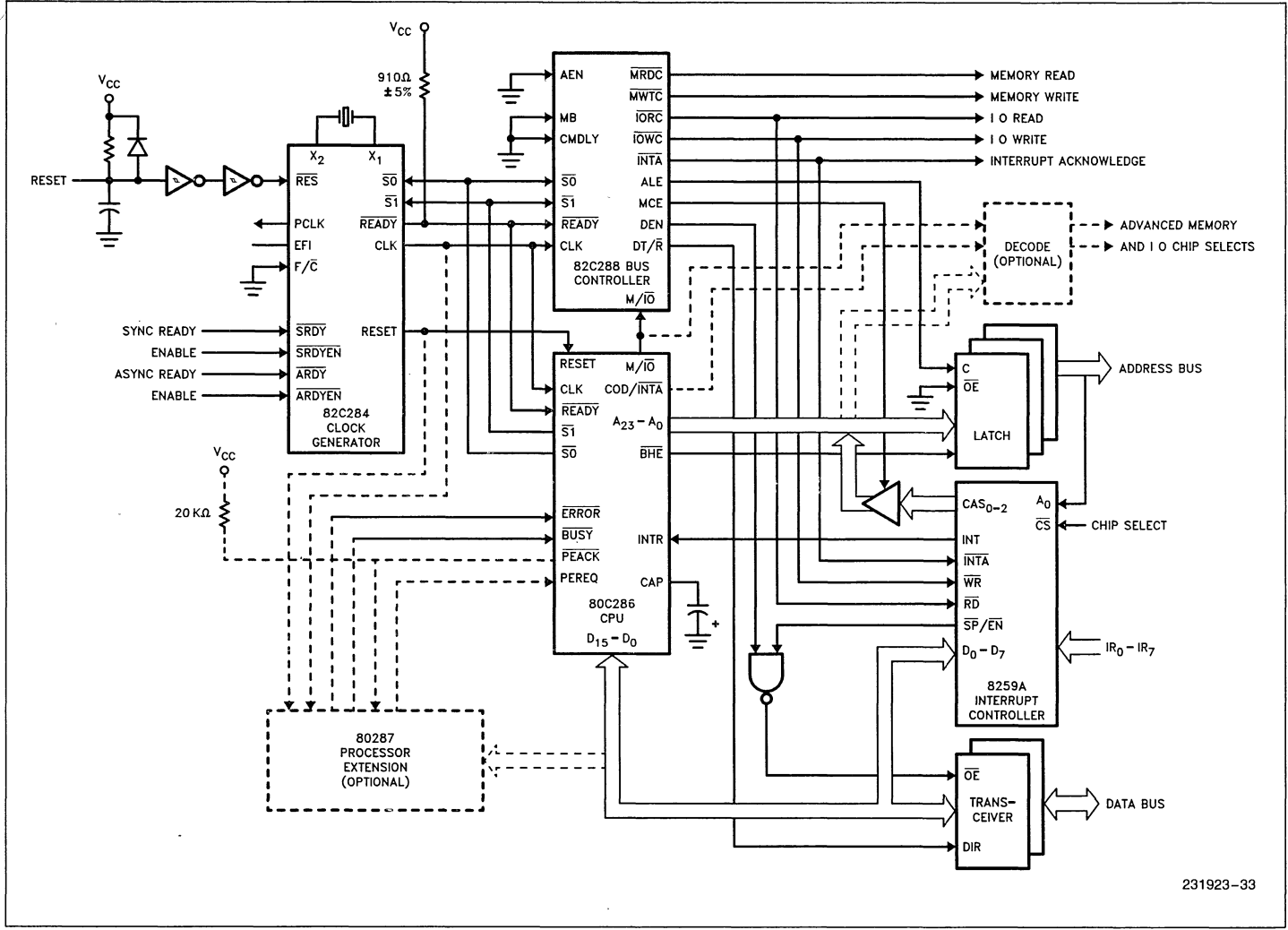


Figure 32. Basic 80C286 System Configuration

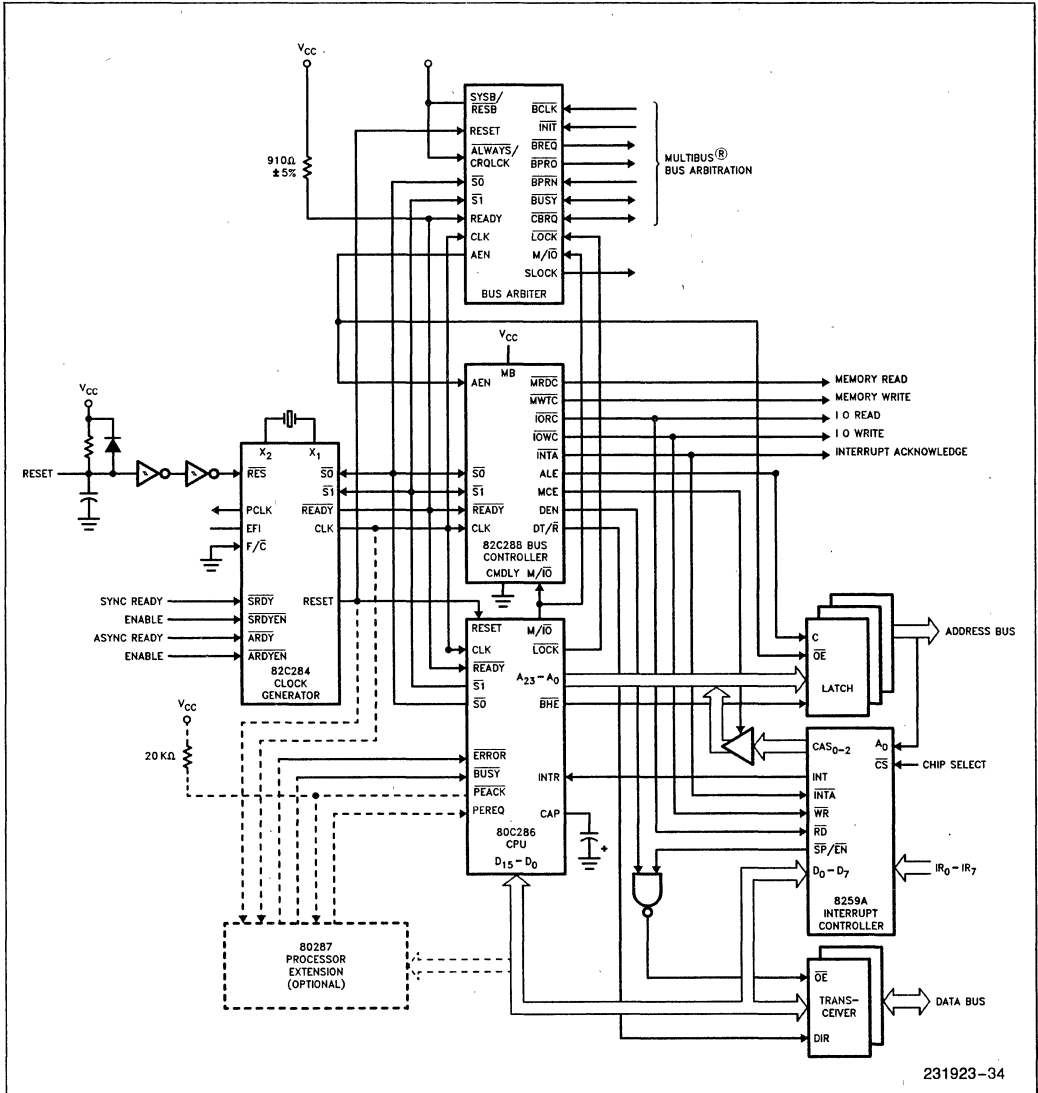


Figure 33. MULTIBUS® System Bus Interface

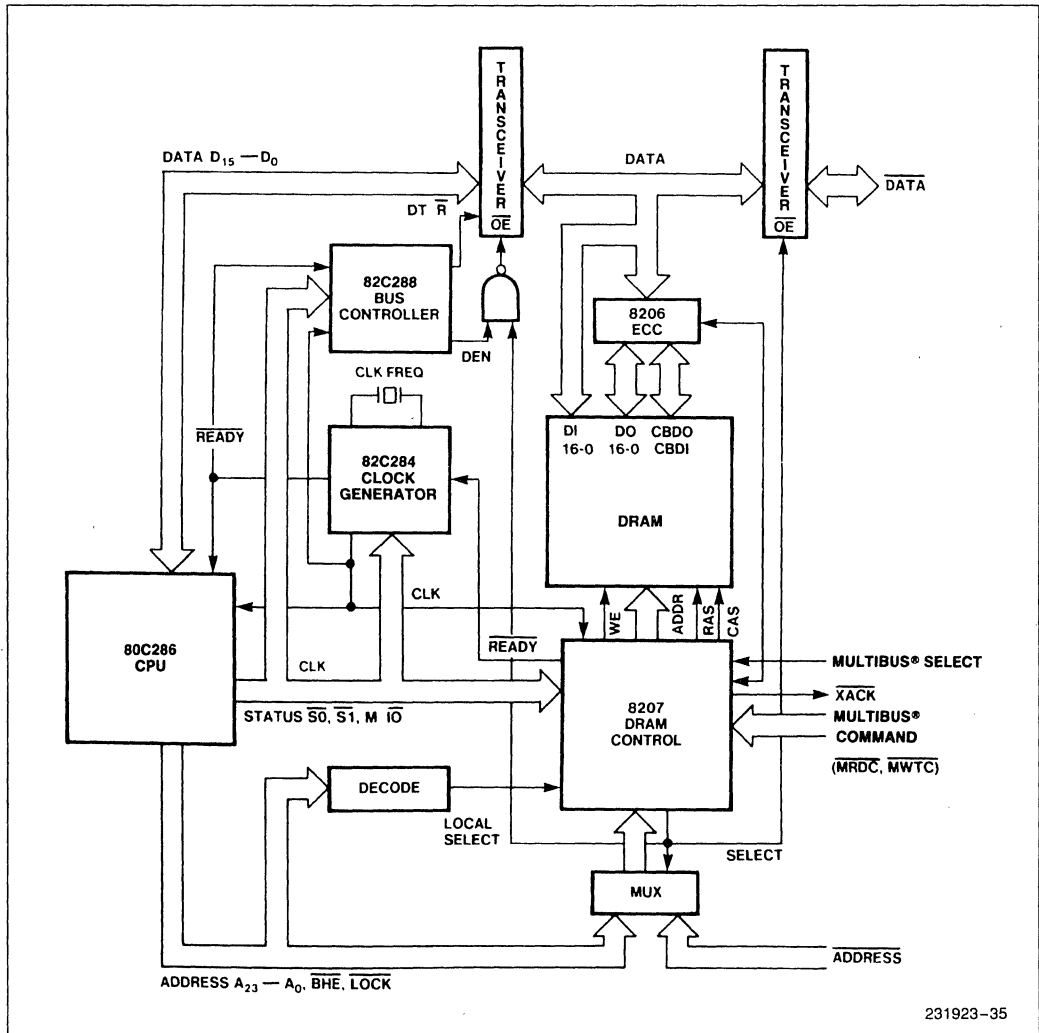


Figure 34. 80C286 System Configuration with Dual-Ported Memory

Figure 34 shows the addition of dual ported dynamic memory between the MULTIBUS system bus and the 80C286 local bus. The dual port interface is provided by the 8207 Dual Port DRAM Controller. The 8207 runs synchronously with the CPU to maximize throughput for local memory references. It also arbitrates between requests from the local and system buses and performs functions such as refresh,

initialization of RAM, and read/modify/write cycles. The 8207 combined with the 8206 Error Checking and Correction memory controller provide for single bit error correction. The dual-ported memory can be combined with a standard MULTIBUS system bus interface to maximize performance and protection in multiprocessor system configurations.

Table 16. 80C286 Systems Recommended Pull Up Resistor Values

80C286 Pin and Name	Pullup Value	Purpose
4— $\overline{S1}$	20 K Ω \pm 10%	Pull $\overline{S0}$, $\overline{S1}$, and \overline{PEACK} inactive during 80C286 hold periods (Note 1)
5— $\overline{S0}$		
6— \overline{PEACK}		
63— \overline{READY}	910 Ω \pm 5%	Pull \overline{READY} inactive within required minimum time ($C_L = 150$ pF, $I_R \leq 7$ mA)

NOTE:

1. Pullup resistors are not required for $\overline{S0}$ and $\overline{S1}$ when the corresponding pins on the 82C284 are connected to $\overline{S0}$ and $\overline{S1}$.

80C286 IN-CIRCUIT EMULATION CONSIDERATIONS

One of the advantages of using the 80C286 is that full in-circuit emulation development support is available through either the I²ICE 80286 probe for 8 MHz/10 MHz or ICE286 for 12.5 MHz designs. To utilize these powerful tools it is necessary that the designer be aware of a few minor parametric and functional differences between the 80C286 and the in-circuit emulators. The I²ICE datasheet (I²ICE Integrated Instrumentation and In-Circuit Emulation System, order #210469) contains a detailed description of these design considerations. The ICE286 Fact Sheet (#280718) and User's Guide (#452317) contain design considerations for the 80C286 12.5 MHz microprocessor. It is recommended that the appropriate document be reviewed by the 80C286 system designer to determine whether or not these differences affect the design.

PACKAGE THERMAL SPECIFICATIONS

The 80C286 Microprocessor is specified for operation when case temperature (T_C) is within the range of 0°C–85°C. Case temperature, unlike ambient temperature, is easily measured in any environment

to determine whether the 80C286 Microprocessor is within the specified operating range. The case temperature should be measured at the center of the top surface of the component.

The maximum ambient temperature (T_A) allowable without violating T_C specifications can be calculated from the equations shown below. T_J is the 80C286 junction temperature. P is the power dissipated by the 80C286.

$$T_J = T_C + P * \theta_{JC}$$

$$T_A = T_J + P * \theta_{JA}$$

$$T_C = T_A + P * [\theta_{JA} - \theta_{JC}]$$

Values for θ_{JA} and θ_{JC} are given in Table 17. θ_{JA} is given at various airflows. Table 18 shows the maximum T_A allowable (without exceeding T_C) at various airflows. Note that the 80C286 PLCC package has an internal heat spreader. T_A can be further improved by attaching "fins" or an external "heat sink" to the package.

Junction temperature calculations should use an I_{CC} value that is measured without external resistive loads. The external resistive loads dissipate additional power external to the 80C286 and not on the die. This increases the resistor temperature, not the die temperature. The full capacitive load ($C_L = 100$ pF) should be applied during the I_{CC} measurement.

Table 17. Thermal Resistances ($^{\circ}$ C/Watt) θ_{JC} and θ_{JA}

Package	θ_{JC}	θ_{JA} versus Airflow ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
68-Lead PGA	5.5	29	22	16	15	14	13
68-Lead PLCC w/Internal Heat Spreader	8	29	23	21	18	16	15

Table 18. Maximum T_A at Various Airflows

Package	$T_A(^{\circ}$ C) versus Airflow ft/min (m/sec)					
	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
68-Lead PGA	67	73	76	78	79	79
68 Lead-PLCC w/Internal Heat Spreader	69	74	75	77	79	80

NOTE:

The numbers in Table 18 were calculated using an I_{CC} of 150 mA, which is representative of the worst case I_{CC} at $T_C = 85^{\circ}$ C with the outputs unloaded.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias 0°C to +70°C
 Storage Temperature -65°C to +150°C
 Voltage on Any Pin with
 Respect to Ground -1.0V to +7V
 Power Dissipation 1.1W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS ($V_{CC} = 5V \pm 10\%$, $T_{CASE} = 0^\circ\text{C}$ to $+85^\circ\text{C}$)

Symbol	Parameter	Min	Max	Typ	Unit	Test Conditions
I_{CC}	Supply Current		200	125	mA	$C_L = 100\text{ pF}$ (Note 1)
I_{CCS}	Supply Current (Static)		5	0.5	mA	(Note 2)
C_{CLK}	CLK Input Capacitance		20		pF	FREQ = 1 MHz (Note 3)
C_{IN}	Other Input Capacitance		10		pF	FREQ = 1 MHz (Note 3)
C_O	Input/Output Capacitance		20		pF	FREQ = 1 MHz (Note 3)

NOTES:

1. Tested at maximum frequency with no resistive loads on the outputs.
2. Tested while clock stopped in phase 2 and inputs at V_{CC} or V_{SS} with the outputs unloaded.
3. These are not tested but are guaranteed by design characterization.

D.C. CHARACTERISTICS ($V_{CC} = 5V \pm 10\%$, $T_{CASE} = 0^\circ\text{C}$ to $+85^\circ\text{C}$)

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL}	Input LOW Voltage	-0.5	0.8	V	FREQ = 2 MHz
V_{IH}	Input HIGH Voltage	2.0	$V_{CC} + 0.5$	V	FREQ = 2 MHz
V_{ILC}	CLK Input LOW Voltage	-0.5	0.8	V	FREQ = 2 MHz
V_{IHC}	CLK Input HIGH Voltage	3.8	$V_{CC} + 0.5$	V	FREQ = 2 MHz
V_{OL}	Output LOW Voltage		0.45	V	$I_{OL} = 2.0\text{ mA}$, FREQ = 2 MHz
V_{OH}	Output HIGH Voltage	3.0 $V_{CC} - 0.5$		V V	$I_{OH} = -2.0\text{ mA}$, FREQ = 2 MHz $I_{OH} = -100\text{ }\mu\text{A}$, FREQ = 2 MHz
I_{LI}	Input Leakage Current		± 10	μA	$V_{IN} = \text{GND or } V_{CC}$ (Note 1)
I_{LO}	Output Leakage Current		± 10	μA	$V_O = \text{GND or } V_{CC}$ (Note 1)
I_{IL}	Input Sustaining Current on BUSY# and ERROR# Pins	-30	-500	μA	$V_{IN} = 0\text{V}$ (Note 1)
I_{BHL}	Input Sustaining Current (Bus Hold LOW)	38	150	μA	$V_{IN} = 1.0\text{V}$ (Notes 1, 2)
I_{BHH}	Input Sustaining Current (Bus Hold HIGH)	-50	-350	μA	$V_{IN} = 3.0\text{V}$ (Notes 1, 3)
I_{BHLO}	Bus Hold LOW Overdrive	200		μA	(Notes 1, 4)
I_{BHHO}	Bus Hold HIGH Overdrive	-400		μA	(Notes 1, 5)

NOTES:

1. Tested with the clock stopped.
2. I_{BHL} should be measured after lowering V_{IN} to GND and then raising to 1.0V on the following pins: 36-51, 66, 67.
3. I_{BHH} should be measured after raising V_{IN} to V_{CC} and then lowering to 3.0V on the following pins: 4-6, 36-51, 66-68.
4. An external driver must source at least I_{BHLO} to switch this node from LOW to HIGH.
5. An external driver must sink at least I_{BHHO} to switch this node from HIGH to LOW.

A.C. CHARACTERISTICS ($V_{CC} = 5V \pm 10\%$, $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$)

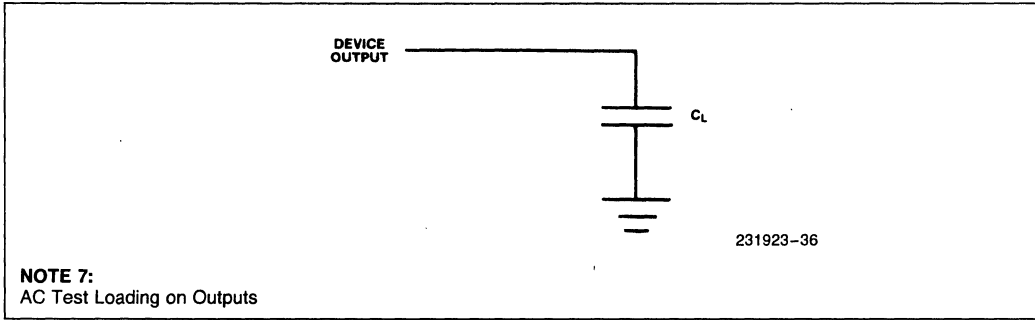
A.C. timings are referenced to 1.5V points of signals as illustrated in datasheet waveforms, unless otherwise noted.

Symbol	Parameter	12.5 MHz		Unit	Test Conditions
		Min	Max		
1	System Clock (CLK) Period	40	DC	ns	(Note 1)
2	System Clock (CLK) LOW Time	11		ns	at 1.0V
3	System Clock (CLK) HIGH Time	13		ns	at 3.6V
17	System Clock (CLK) Rise Time		8	ns	1.0V to 3.6V (Note 2)
18	System Clock (CLK) Fall Time		8	ns	3.6V to 1.0V (Note 2)
4	Asynchronous Inputs Setup Time	16		ns	(Note 3)
5	Asynchronous Inputs Hold Time	16		ns	(Note 3)
6	RESET Setup Time	19		ns	
7	RESET Hold Time	6		ns	
8	Read Data Setup Time	6		ns	
9	Read Data Hold Time	7		ns	
10	\overline{READY} Setup Time	23		ns	
11	\overline{READY} Hold Time	21		ns	
12a1	Status Active Delay	5	16	ns	(Notes 4, 5)
12a2	\overline{PEACK} Active Delay	5	18	ns	(Notes 4, 5)
12b	Status/ \overline{PEACK} Inactive Delay	5	20	ns	(Notes 4, 5)
13	Address Valid Delay	4	29	ns	(Notes 4, 5)
14	Write Data Valid Delay	3	27	ns	(Notes 4, 5)
15	Address/Status/Data Float Delay	2	32	ns	(Notes 2, 4, 6)
16	HLDA Valid Delay	3	24	ns	(Notes 4, 5)
19	Address Valid To Status Valid Setup Time	23		ns	(Notes 2, 4, 5)

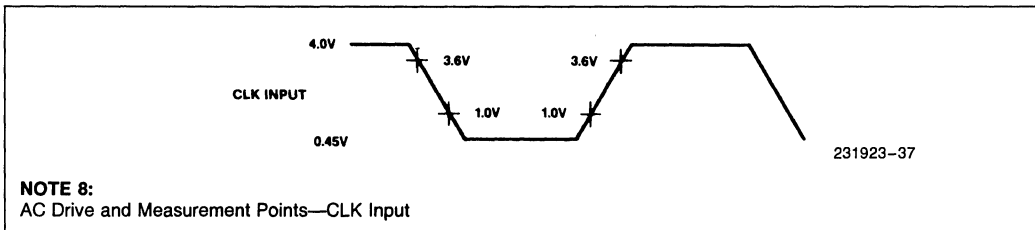
NOTES:

- Functionality at frequencies less than 2 MHz is not tested, but is guaranteed by design characterization.
- These are not tested but are guaranteed by design characterization.
- Asynchronous inputs are INTR, NMI, HOLD, PEREQ, ERROR, and BUSY. This specification is given only for testing purposes, to assure recognition at a specific CLK edge.
- Delay from 1.0V on the CLK, to 1.5V or float on the output as appropriate for valid or floating condition.
- Output load: $C_L = 100$ pF.
- Float condition occurs when output current is less than I_{LO} in magnitude.

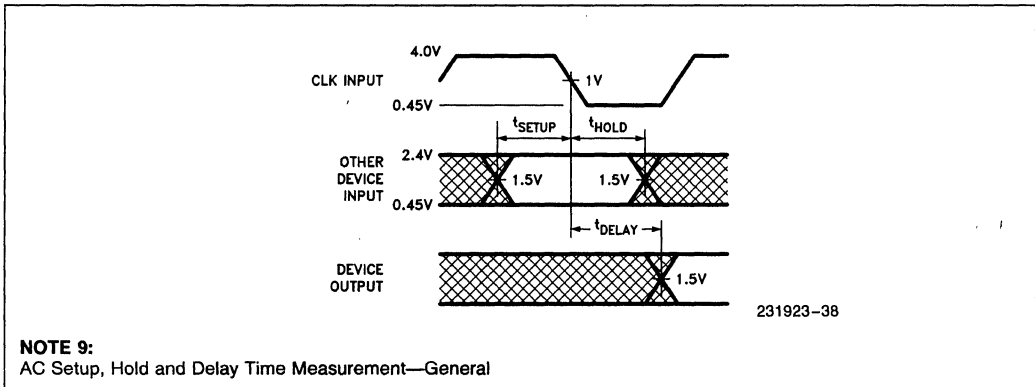
A.C. CHARACTERISTICS (Continued)



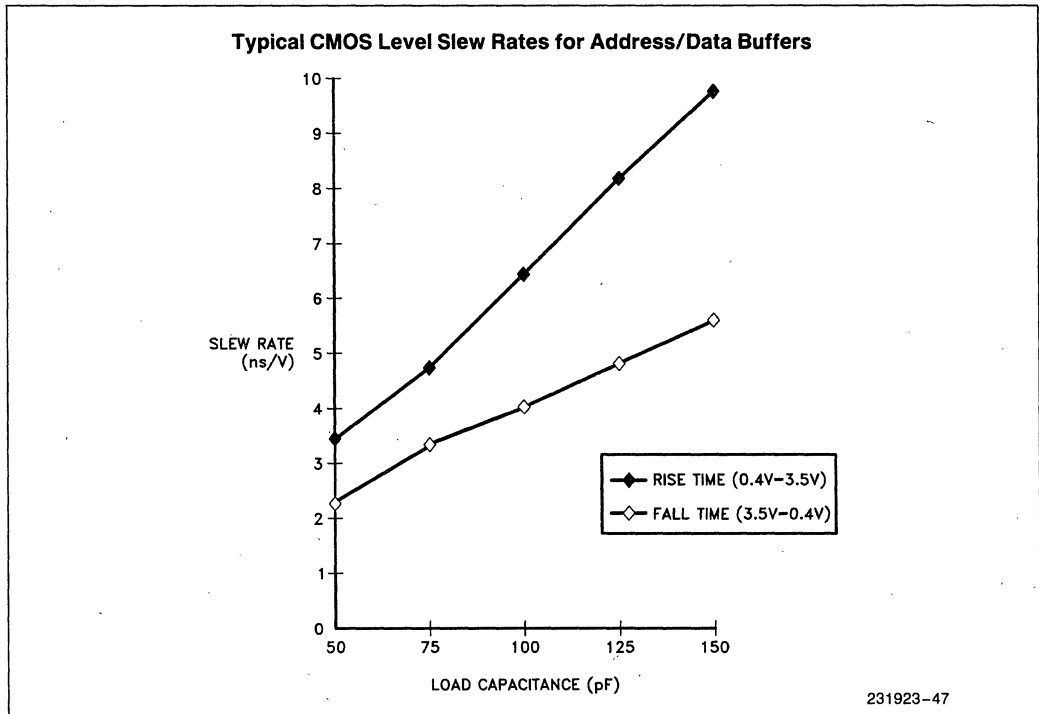
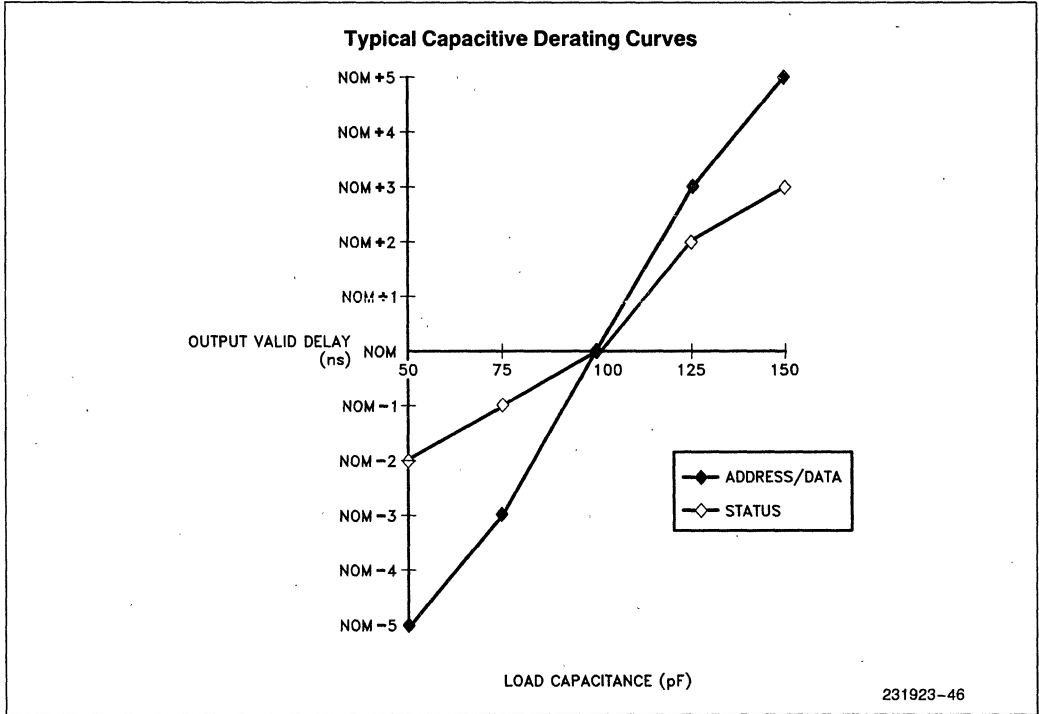
NOTE 7:
AC Test Loading on Outputs



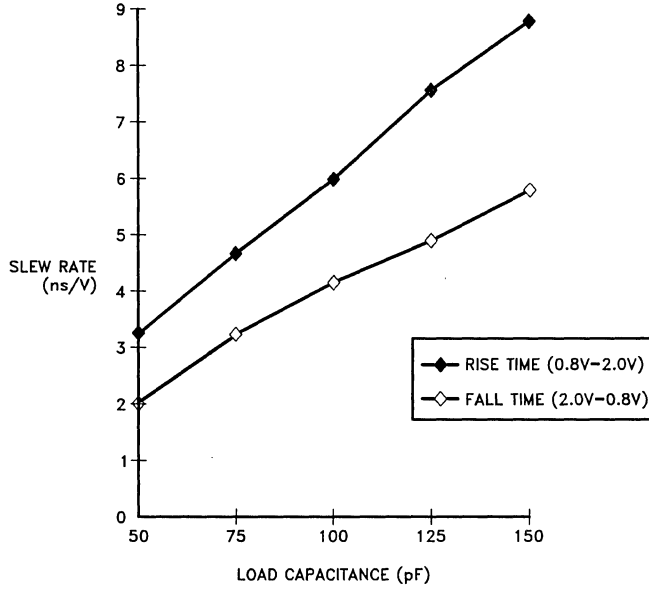
NOTE 8:
AC Drive and Measurement Points—CLK Input



NOTE 9:
AC Setup, Hold and Delay Time Measurement—General

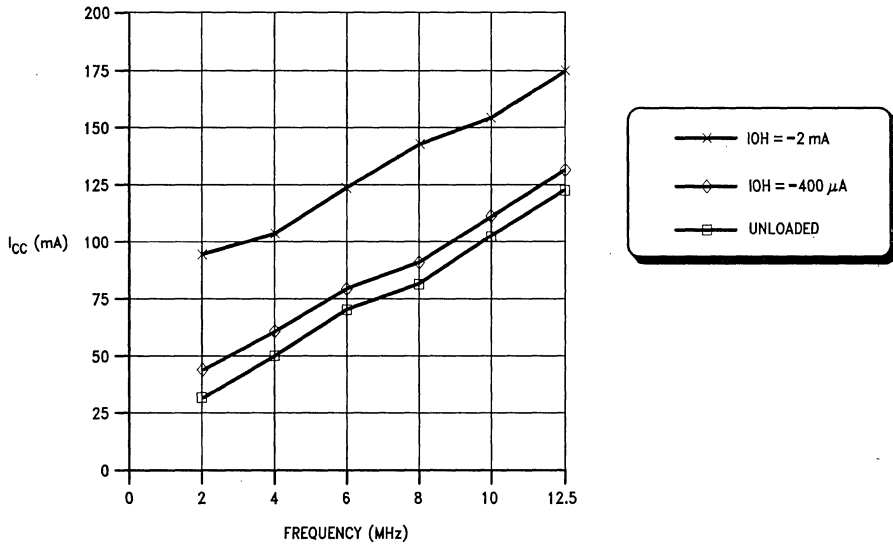


Typical TTL Level Slew Rates for Address/Data Buffers



231923-48

Typical I_{CC} vs Frequency for Different Output Loads



231923-53

NOTES:

1. $V_{CC} = 5.0V$
 2. Loaded: $I_{OL} = 2.0$ mA, I_{OH} as shown, $C_L = 100$ pF
- Unloaded: $C_L = 100$ pF

A.C. CHARACTERISTICS (Continued)**82C284 Timing Requirements**

Symbol	Parameter	82C284-12		Unit	Test Conditions
		Min	Max		
11	$\overline{\text{SRDY}}/\overline{\text{SRDYEN}}$ Setup Time	18		ns	
12	$\overline{\text{SRDY}}/\overline{\text{SRDYEN}}$ Hold Time	2		ns	
13	$\overline{\text{ARDY}}/\overline{\text{ARDYEN}}$ Setup Time	0		ns	(Note 1)
14	$\overline{\text{ARDY}}/\overline{\text{ARDYEN}}$ Hold Time	25		ns	(Note 1)
19	PCLK Delay	0	23	ns	$C_L = 75 \text{ pF}$ $I_{OL} = 5 \text{ mA}$ $I_{OH} = -1 \text{ mA}$

NOTE:

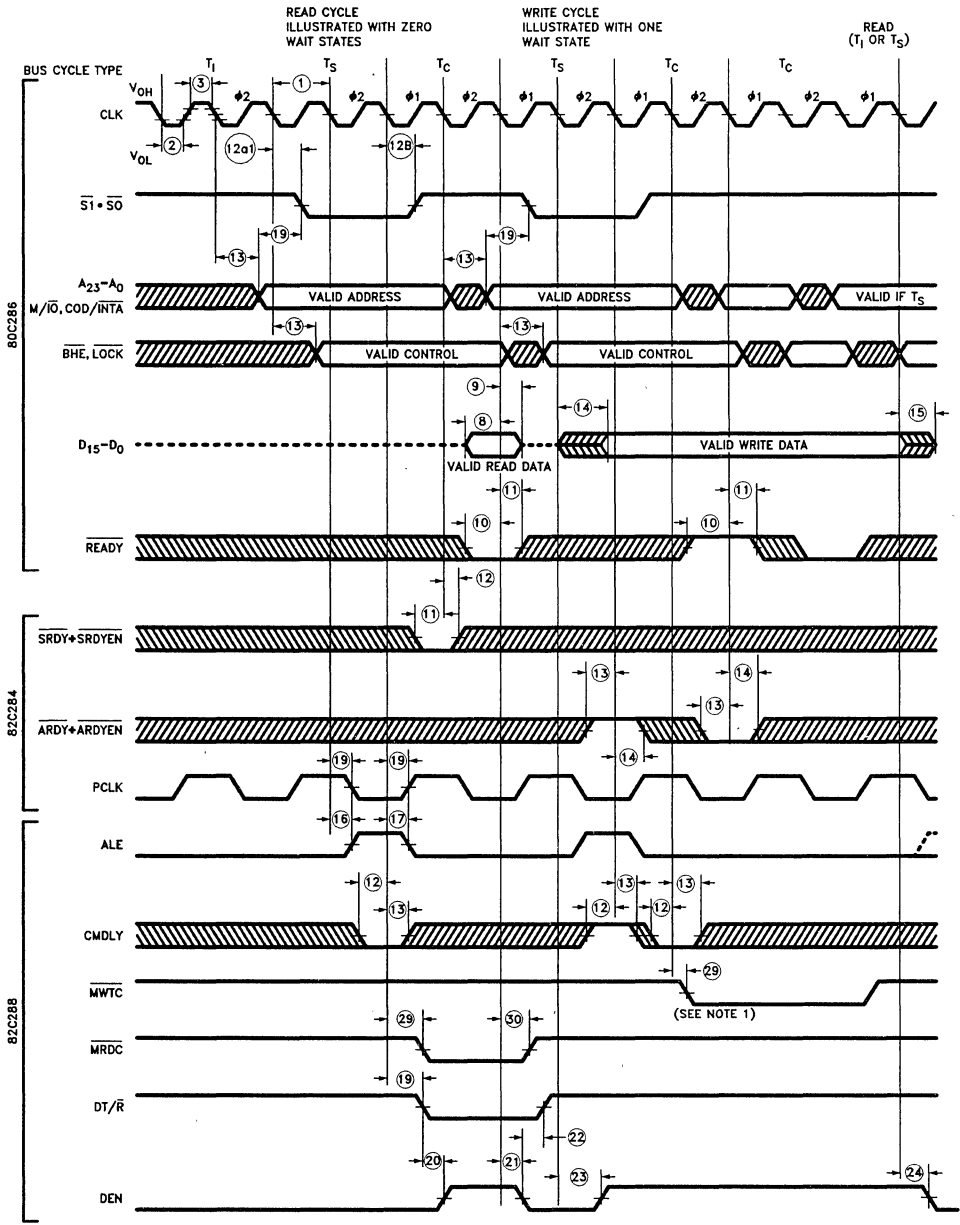
1. These times are given for testing purposes to assure a predetermined action.

82C288 Timing Requirements

Symbol	Parameter		82C288-12		Unit	Test Conditions
			Min	Max		
12	CMDLY Setup Time		15		ns	
13	CMDLY Hold Time		1		ns	
30	Command Delay from CLK	Command Inactive	5	20	ns	$C_L = 300 \text{ pF max}$ $I_{OL} = 32 \text{ mA max}$ $I_{OH} = -5 \text{ mA max}$
29		Command Active	3	21		
16	ALE Active Delay		3	16	ns	$C_L = 150 \text{ pF}$ $I_{OL} = 16 \text{ mA max}$ $I_{OH} = -1 \text{ mA max}$
17	ALE Inactive Delay			19	ns	
19	DT/ $\overline{\text{R}}$ Read Active Delay			23	ns	
22	DT/ $\overline{\text{R}}$ Read Inactive Delay		5	18	ns	
20	DEN Read Active Delay		5	21	ns	
21	DEN Read Inactive Delay		3	19	ns	
23	DEN Write Active Delay			23	ns	
24	DEN Write Inactive Delay		3	19	ns	

WAVEFORMS

MAJOR CYCLE TIMING

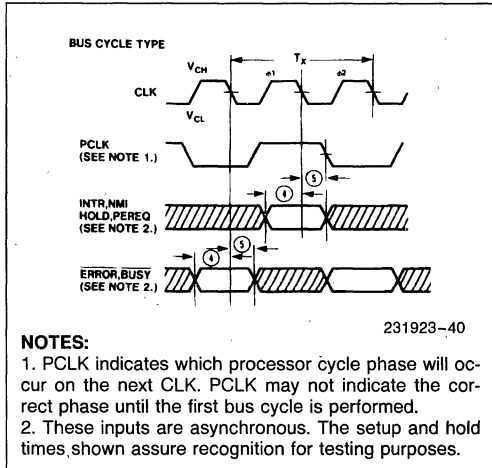


231923-52

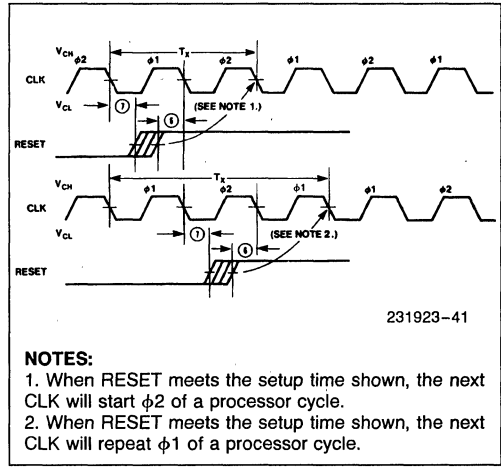
NOTE:
1. The modified timing is due to the $\overline{\text{CMDLY}}$ signal being active.

WAVEFORMS (Continued)

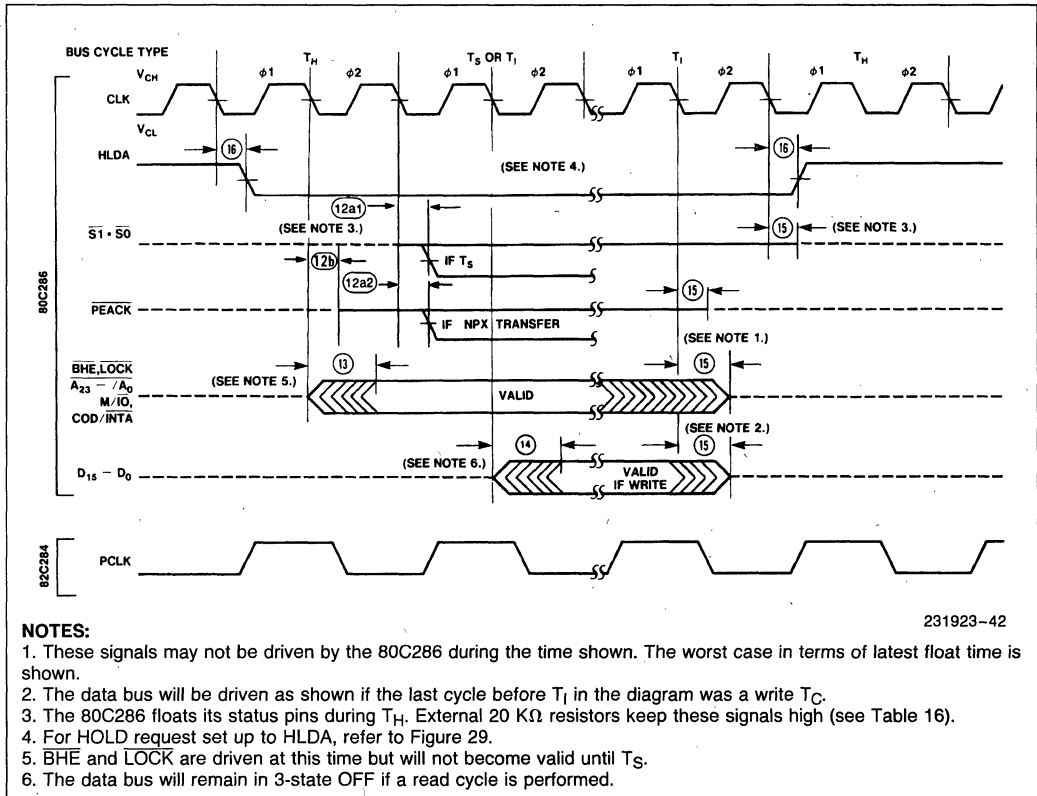
80C286 ASYNCHRONOUS INPUT SIGNAL TIMING



80C286 RESET INPUT TIMING AND SUBSEQUENT PROCESSOR CYCLE PHASE

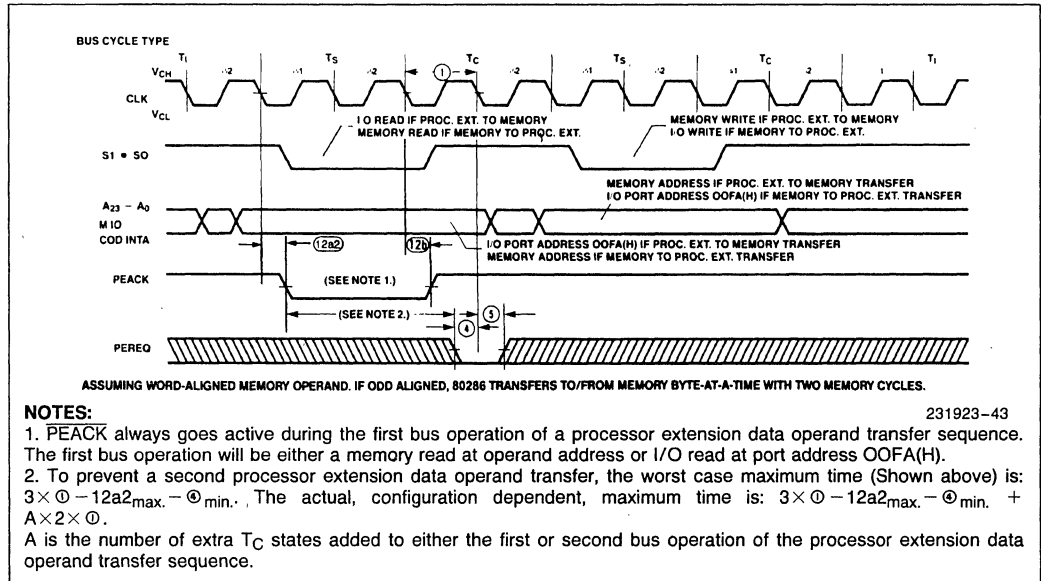


EXITING AND ENTERING HOLD

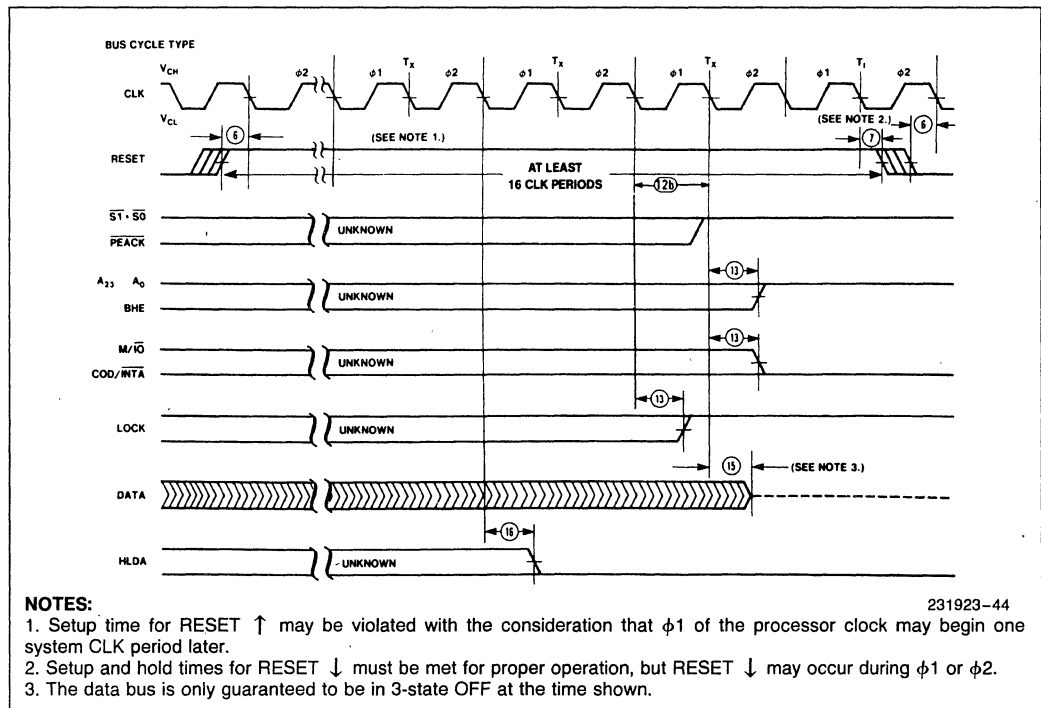


WAVEFORMS (Continued)

80C286 PEREQ/PEACK TIMING FOR ONE TRANSFER ONLY



INITIAL 80C286 PIN STATE DURING RESET



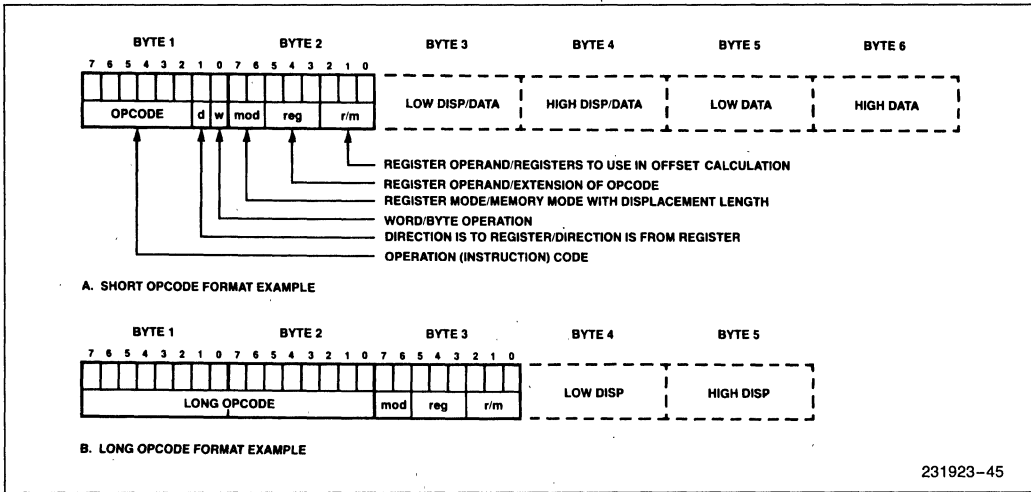


Figure 35. 80C286 Instruction Format Examples

80C286 INSTRUCTION SET SUMMARY

Instruction Timing Notes

The instruction clock counts listed below establish the maximum execution rate of the 80C286. With no delays in bus cycles, the actual clock count of an 80C286 program will average 5% more than the calculated clock count, due to instruction sequences which execute faster than they can be fetched from memory.

To calculate elapsed times for instruction sequences, multiply the sum of all instruction clock counts, as listed in the table below, by the processor clock period. A 12 MHz processor clock has a clock period of 83 nanoseconds and requires an 80C286 system clock (CLK input) of 24 MHz.

Instruction Clock Count Assumptions

1. The instruction has been prefetched, decoded, and is ready for execution. Control transfer instruction clock counts include all time required to fetch, decode, and prepare the next instruction for execution.
2. Bus cycles do not require wait states.
3. There are no processor extension data transfer or local bus HOLD requests.
4. No exceptions occur during instruction execution.

Instruction Set Summary Notes

Addressing displacements selected by the MOD field are not shown. If necessary they appear after the instruction fields shown.

- Above/below refers to unsigned value
- Greater refers to positive signed value
- Less refers to less positive (more negative) signed values

- if d = 1 then to register; if d = 0 then from register
- if w = 1 then word instruction; if w = 0 then byte instruction
- if s = 0 then 16-bit immediate data form the operand
- if s = 1 then an immediate data byte is sign-extended to form the 16-bit operand

- x don't care
- z used for string primitives for comparison with ZF FLAG

If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand

- * = add one clock if offset calculation requires summing 3 elements
- n = number of times repeated
- m = number of bytes of code in next instruction
- Level (L)—Lexical nesting level of the procedure

The following comments describe possible exceptions, side effects, and allowed usage for instructions in both operating modes of the 80C286.

REAL ADDRESS MODE ONLY

1. This is a protected mode instruction. Attempted execution in real address mode will result in an undefined opcode exception (6).
2. A segment overrun exception (13) will occur if a word operand reference at offset FFFF(H) is attempted.
3. This instruction may be executed in real address mode to initialize the CPU for protected mode.
4. The IOPL and NT fields will remain 0.
5. Processor extension segment overrun interrupt (9) will occur if the operand exceeds the segment limit.

EITHER MODE

6. An exception may occur, depending on the value of the operand.
7. $\overline{\text{LOCK}}$ is automatically asserted regardless of the presence or absence of the LOCK instruction prefix.
8. $\overline{\text{LOCK}}$ does not remain active between all operand transfers.

PROTECTED VIRTUAL ADDRESS MODE ONLY

9. A general protection exception (13) will occur if the memory operand cannot be used due to either a segment limit or access rights violation. If a stack segment limit is violated, a stack segment overrun exception (12) occurs.
10. For segment load operations, the CPL, RPL, and DPL must agree with privilege rules to avoid an exception. The segment must be present to

avoid a not-present exception (11). If the SS register is the destination, and a segment not-present violation occurs, a stack exception (12) occurs.

11. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert LOCK to maintain descriptor integrity in multiprocessor systems.
12. JMP, CALL, INT, RET, IRET instructions referring to another code segment will cause a general protection exception (13) if any privilege rule is violated.
13. A general protection exception (13) occurs if $\text{CPL} \neq 0$.
14. A general protection exception (13) occurs if $\text{CPL} > \text{IOPL}$.
15. The IF field of the flag word is not updated if $\text{CPL} > \text{IOPL}$. The IOPL field is updated only if $\text{CPL} = 0$.
16. Any violation of privilege rules as applied to the selector operand do not cause a protection exception; rather, the instruction does not return a result and the zero flag is cleared.
17. If the starting address of the memory operand violates a segment limit, or an invalid access is attempted, a general protection exception (13) will occur before the ESC instruction is executed. A stack segment overrun exception (12) will occur if the stack limit is violated by the operand's starting address. If a segment limit is violated during an attempted data transfer then a processor extension segment overrun exception (9) occurs.
18. The destination of an INT, JMP, CALL, RET or IRET instruction must be in the defined limit of a code segment or a general protection exception (13) will occur.

80C286 INSTRUCTION SET SUMMARY

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
DATA TRANSFER					
MOV = Move:					
Register to Register/Memory	1 0 0 0 1 0 0 w mod reg r/m	2,3*	2,3*	2	9
Register/memory to register	1 0 0 0 1 0 1 w mod reg r/m	2,5*	2,5*	2	9
Immediate to register/memory	1 1 0 0 0 1 1 w mod 0 0 0 r/m data data if w = 1	2,3*	2,3*	2	9
Immediate to register	1 0 1 1 w reg data data if w = 1	2	2		
Memory to accumulator	1 0 1 0 0 0 0 w addr-low addr-high	5	5	2	9
Accumulator to memory	1 0 1 0 0 0 1 w addr-low addr-high	3	3	2	9
Register/memory to segment register	1 0 0 0 1 1 1 0 mod 0 reg r/m	2,5*	17,19*	2	9,10,11
Segment register to register/memory	1 0 0 0 1 1 0 0 mod 0 reg r/m	2,3*	2,3*	2	9
PUSH = Push:					
Memory	1 1 1 1 1 1 1 1 mod 1 1 0 r/m	5*	5*	2	9
Register	0 1 0 1 0 reg	3	3	2	9
Segment register	0 0 0 reg 1 1 0	3	3	2	9
Immediate	0 1 1 0 1 0 s 0 data data if s = 0	3	3	2	9
PUSHA = Push All	0 1 1 0 0 0 0 0	17	17	2	9
POP = Pop:					
Memory	1 0 0 0 1 1 1 1 mod 0 0 0 r/m	5*	5*	2	9
Register	0 1 0 1 1 reg	5	5	2	9
Segment register	0 0 0 reg 1 1 1 (reg ≠ 01)	5	20	2	9,10,11
POPA = Pop All	0 1 1 0 0 0 0 1	19	19	2	9
XCHG = Exchange:					
Register/memory with register	1 0 0 0 0 1 1 w mod reg r/m	3,5*	3,5*	2,7	7,9
Register with accumulator	1 0 0 1 0 reg	3	3		
IN = Input from:					
Fixed port	1 1 1 0 0 1 0 w port	5	5		14
Variable port	1 1 1 0 1 1 0 w	5	5		14
OUT = Output to:					
Fixed port	1 1 1 0 0 1 1 w port	3	3		14
Variable port	1 1 1 0 1 1 1 w	3	3		14
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1	5	5		9
LEA = Load EA to register	1 0 0 0 1 1 0 1 mod reg r/m	3*	3*		
LDS = Load pointer to DS	1 1 0 0 0 1 0 1 mod reg r/m (mod ≠ 11)	7*	21*	2	9,10,11
LES = Load pointer to ES	1 1 0 0 0 1 0 0 mod reg r/m (mod ≠ 11)	7*	21*	2	9,10,11

Shaded areas indicate instructions not available in 8086, 88 microsystems.

80C286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
DATA TRANSFER (Continued)					
LAHF Load AH with flags	1 0 0 1 1 1 1 1	2	2		
SAHF = Store AH into flags	1 0 0 1 1 1 1 0	2	2		
PUSHF = Push flags	1 0 0 1 1 1 0 0	3	3	2	9
POPF = Pop flags	1 0 0 1 1 1 0 1	5	5	2,4	9,15
ARITHMETIC					
ADD = Add:					
Reg/memory with register to either	0 0 0 0 0 0 d w mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 s w mod 0 0 0 r/m data data if s w = 01	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 0 0 1 0 w data data if w = 1	3	3		
ADC = Add with carry:					
Reg/memory with register to either	0 0 0 1 0 0 d w mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 s w mod 0 1 0 r/m data data if s w = 01	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 1 0 1 0 w data data if w = 1	3	3		
INC = Increment:					
Register/memory	1 1 1 1 1 1 1 w mod 0 0 0 r/m	2,7*	2,7*	2	9
Register	0 1 0 0 0 reg	2	2		
SUB = Subtract:					
Reg/memory and register to either	0 0 1 0 1 0 d w mod reg r/m	2,7*	2,7*	2	9
Immediate from register/memory	1 0 0 0 0 0 s w mod 1 0 1 r/m data data if s w = 01	3,7*	3,7*	2	9
Immediate from accumulator	0 0 1 0 1 1 0 w data data if w = 1	3	3		
SBB = Subtract with borrow:					
Reg/memory and register to either	0 0 0 1 1 0 d w mod reg r/m	2,7*	2,7*	2	9
Immediate from register/memory	1 0 0 0 0 0 s w mod 0 1 1 r/m data data if s w = 01	3,7*	3,7*	2	9
Immediate from accumulator	0 0 0 1 1 1 0 w data data if w = 1	3	3		
DEC = Decrement					
Register/memory	1 1 1 1 1 1 1 w mod 0 0 1 r/m	2,7*	2,7*	2	9
Register	0 1 0 0 1 reg	2	2		
CMP = Compare					
Register/memory with register	0 0 1 1 1 0 1 w mod reg r/m	2,6*	2,6*	2	9
Register with register/memory	0 0 1 1 1 0 0 w mod reg r/m	2,7*	2,7*	2	9
Immediate with register/memory	1 0 0 0 0 0 s w mod 1 1 1 r/m data data if s w = 01	3,6*	3,6*	2	9
Immediate with accumulator	0 0 1 1 1 1 0 w data data if w = 1	3	3		
NEG = Change sign	1 1 1 1 0 1 1 w mod 0 1 1 r/m	2	7*	2	9
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1	3	3		
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1	3	3		

80C286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
ARITHMETIC (Continued)					
AAS = ASCII adjust for subtract	00111111	3	3		
DAS = Decimal adjust for subtract	00101111	3	3		
MUL = Multiply (unsigned):	1111011w mod 100 r/m				
Register-Byte		13	13		
Register-Word		21	21		
Memory-Byte		16*	16*	2	9
Memory-Word		24*	24*	2	9
IMUL = Integer multiply (signed):	1111011w mod 101 r/m				
Register-Byte		13	13		
Register-Word		21	21		
Memory-Byte		16*	16*	2	9
Memory-Word		24*	24*	2	9
IMUL = Integer immediate multiply (signed)	011010s1 mod reg r/m data data if s = 0	21,24*	21,24*	2	9
DIV = Divide (unsigned)	1111011w mod 110 r/m				
Register-Byte		14	14	6	6
Register-Word		22	22	6	6
Memory-Byte		17*	17*	2,6	6,9
Memory-Word		25*	25*	2,6	6,9
IDIV = Integer divide (signed)	1111011w mod 111 r/m				
Register-Byte		17	17	6	6
Register-Word		25	25	6	6
Memory-Byte		20*	20*	2,6	6,9
Memory-Word		28*	28*	2,6	6,9
AAM = ASCII adjust for multiply	11010100 00001010	16	16		
AAD = ASCII adjust for divide	11010101 00001010	14	14		
CBW = Convert byte to word	10011000	2	2		
CWD = Convert word to double word	10011001	2	2		
LOGIC					
Shift/Rotate Instructions:					
Register/Memory by 1	1101000w mod TTT r/m	2,7*	2,7*	2	9
Register/Memory by CL	1101001w mod TTT r/m	5+n,8+n*	5+n,8+n*	2	9
Register/Memory by Count	1100000w mod TTT r/m count	5+n,8+n*	5+n,8+n*	2	9
	TTT Instruction				
	000 ROL				
	001 ROR				
	010 RCL				
	011 RCR				
	100 SHL/SAL				
	101 SHR				
	111 SAR				

Shaded areas indicate instructions not available in 8086, 88 microsystems.

80C286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
ARITHMETIC (Continued)					
AND = And:					
Reg/memory and register to either	001000dw mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1000000w mod 100 r/m data data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0010010w data data if w = 1	3	3		
TEST = And function to flags, no result:					
Register/memory and register	1000010w mod reg r/m	2,6*	2,6*	2	9
Immediate data and register/memory	1111011w mod 000 r/m data data if w = 1	3,6*	3,6*	2	9
Immediate data and accumulator	1010100w data data if w = 1	3	3		
OR = Or:					
Reg/memory and register to either	000010dw mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1000000w mod 001 r/m data data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0000110w data data if w = 1	3	3		
XOR = Exclusive or:					
Reg/memory and register to either	001100dw mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1000000w mod 110 r/m data data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0011010w data data if w = 1	3	3		
NOT = Invert register/memory	1111011w mod 010 r/m	2,7*	2,7*	2	9
STRING MANIPULATION:					
MOVS = Move byte/word	1010010w	5	5	2	9
CMPS = Compare byte/word	1010011w	8	8	2	9
SCAS = Scan byte/word	1010111w	7	7	2	9
LODS = Load byte/wd to AL/AX	1010110w	5	5	2	9
STOS = Stor byte/wd from AL/A	1010101w	3	3	2	9
INS = Input byte/wd from DX port	0110110w	5	5	2	9,14
OUTS = Output byte/wd to DX port	0110111w	5	5	2	9,14
Repeated by count in CX					
MOV₅ = Move string	11110011 1010010w	5+4n	5+4n	2	9
CMPS = Compare string	1111001z 1010011w	5+9n	5+9n	2,8	8,9
SCAS = Scan string	1111001z 1010111w	5+8n	5+8n	2,8	8,9
LODS = Load string	11110011 1010110w	5+4n	5+4n	2,8	8,9
STOS = Store string	11110011 1010101w	4+3n	4+3n	2,8	8,9
INS = Input string	11110011 0110110w	5+4n	5+4n	2	9,14
OUTS = Output string	11110011 0110111w	5+4n	5+4n	2	9,14

Shaded areas indicate instructions not available in 8086, 88 microsystems.

80C286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS					
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode				
CONTROL TRANSFER									
CALL = Call:									
Direct within segment	<table border="1"><tr><td>1 1 1 0 1 0 0 0</td><td>disp-low</td><td>disp-high</td></tr></table>	1 1 1 0 1 0 0 0	disp-low	disp-high	7 + m	7 + m	2	18	
1 1 1 0 1 0 0 0	disp-low	disp-high							
Register/memory indirect within segment	<table border="1"><tr><td>1 1 1 1 1 1 1 1</td><td>mod 0 1 0</td><td>r/m</td></tr></table>	1 1 1 1 1 1 1 1	mod 0 1 0	r/m	7 + m, 11 + m*	7 + m, 11 + m*	2,8	8,9,18	
1 1 1 1 1 1 1 1	mod 0 1 0	r/m							
Direct intersegment	<table border="1"><tr><td>1 0 0 1 1 0 1 0</td><td>segment offset</td></tr></table>	1 0 0 1 1 0 1 0	segment offset	13 + m	26 + m	2	11,12,18		
1 0 0 1 1 0 1 0	segment offset								
Protected Mode Only (Direct intersegment):	<table border="1"><tr><td>segment selector</td></tr></table>	segment selector							
segment selector									
Via call gate to same privilege level			41 + m		8,11,12,18				
Via call gate to different privilege level, no parameters			82 + m		8,11,12,18				
Via call gate to different privilege level, x parameters			86 + 4x + m		8,11,12,18				
Via TSS			177 + m		8,11,12,18				
Via task gate			182 + m		8,11,12,18				
Indirect intersegment	<table border="1"><tr><td>1 1 1 1 1 1 1 1</td><td>mod 0 1 1</td><td>r/m</td><td>(mod ≠ 11)</td></tr></table>	1 1 1 1 1 1 1 1	mod 0 1 1	r/m	(mod ≠ 11)	16 + m	29 + m*	2	8,9,11,12,18
1 1 1 1 1 1 1 1	mod 0 1 1	r/m	(mod ≠ 11)						
Protected Mode Only (Indirect intersegment):									
Via call gate to same privilege level			44 + m*		8,9,11,12,18				
Via call gate to different privilege level, no parameters			83 + m*		8,9,11,12,18				
Via call gate to different privilege level, x parameters			90 + 4x + m*		8,9,11,12,18				
Via TSS			180 + m*		8,9,11,12,18				
Via task gate			185 + m*		8,9,11,12,18				
JMP = Unconditional jump:									
Short/long	<table border="1"><tr><td>1 1 1 0 1 0 1 1</td><td>disp-low</td></tr></table>	1 1 1 0 1 0 1 1	disp-low	7 + m	7 + m		18		
1 1 1 0 1 0 1 1	disp-low								
Direct within segment	<table border="1"><tr><td>1 1 1 0 1 0 0 1</td><td>disp-low</td><td>disp-high</td></tr></table>	1 1 1 0 1 0 0 1	disp-low	disp-high	7 + m	7 + m		18	
1 1 1 0 1 0 0 1	disp-low	disp-high							
Register/memory indirect within segment	<table border="1"><tr><td>1 1 1 1 1 1 1 1</td><td>mod 1 0 0</td><td>r/m</td></tr></table>	1 1 1 1 1 1 1 1	mod 1 0 0	r/m	7 + m, 11 + m*	7 + m, 11 + m*	2	9,18	
1 1 1 1 1 1 1 1	mod 1 0 0	r/m							
Direct intersegment	<table border="1"><tr><td>1 1 1 0 1 0 1 0</td><td>segment offset</td></tr></table>	1 1 1 0 1 0 1 0	segment offset	11 + m	23 + m		11,12,18		
1 1 1 0 1 0 1 0	segment offset								
Protected Mode Only (Direct intersegment):	<table border="1"><tr><td>segment selector</td></tr></table>	segment selector							
segment selector									
Via call gate to same privilege level			38 + m		8,11,12,18				
Via TSS			175 + m		8,11,12,18				
Via task gate			180 + m		8,11,12,18				
Indirect intersegment	<table border="1"><tr><td>1 1 1 1 1 1 1 1</td><td>mod 1 0 1</td><td>r/m</td><td>(mod ≠ 11)</td></tr></table>	1 1 1 1 1 1 1 1	mod 1 0 1	r/m	(mod ≠ 11)	15 + m*	26 + m*	2	8,9,11,12,18
1 1 1 1 1 1 1 1	mod 1 0 1	r/m	(mod ≠ 11)						
Protected Mode Only (Indirect intersegment):									
Via call gate to same privilege level			41 + m*		8,9,11,12,18				
Via TSS			178 + m*		8,9,11,12,18				
Via task gate			183 + m*		8,9,11,12,18				
RET = Return from CALL:									
Within segment	<table border="1"><tr><td>1 1 0 0 0 0 1 1</td></tr></table>	1 1 0 0 0 0 1 1	11 + m	11 + m	2	8,9,18			
1 1 0 0 0 0 1 1									
Within seg adding immed to SP	<table border="1"><tr><td>1 1 0 0 0 0 1 0</td><td>data-low</td><td>data-high</td></tr></table>	1 1 0 0 0 0 1 0	data-low	data-high	11 + m	11 + m	2	8,9,18	
1 1 0 0 0 0 1 0	data-low	data-high							
Intersegment	<table border="1"><tr><td>1 1 0 0 1 0 1 1</td></tr></table>	1 1 0 0 1 0 1 1	15 + m	25 + m	2	8,9,11,12,18			
1 1 0 0 1 0 1 1									
Intersegment adding immediate to SP	<table border="1"><tr><td>1 1 0 0 1 0 1 0</td><td>data-low</td><td>data-high</td></tr></table>	1 1 0 0 1 0 1 0	data-low	data-high	15 + m		2	8,9,11,12,18	
1 1 0 0 1 0 1 0	data-low	data-high							
Protected Mode Only (RET):									
To different privilege level			55 + m		9,11,12,18				

80C286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS					
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode				
CONTROL TRANSFER (Continued)									
JE/JZ = Jump on equal zero	<table border="1"><tr><td>01110100</td><td>disp</td></tr></table>	01110100	disp	7+m or 3	7+m or 3		18		
01110100	disp								
JL/JNGE = Jump on less/not greater or equal	<table border="1"><tr><td>01111100</td><td>disp</td></tr></table>	01111100	disp	7+m or 3	7+m or 3		18		
01111100	disp								
JLE/JNG = Jump on less or equal/not greater	<table border="1"><tr><td>01111110</td><td>disp</td></tr></table>	01111110	disp	7+m or 3	7+m or 3		18		
01111110	disp								
JB/JNAE = Jump on below/not above or equal	<table border="1"><tr><td>01110010</td><td>disp</td></tr></table>	01110010	disp	7+m or 3	7+m or 3		18		
01110010	disp								
JBE/JNA = Jump on below or equal/not above	<table border="1"><tr><td>01110110</td><td>disp</td></tr></table>	01110110	disp	7+m or 3	7+m or 3		18		
01110110	disp								
JP/JPE = Jump on parity/parity even	<table border="1"><tr><td>01111010</td><td>disp</td></tr></table>	01111010	disp	7+m or 3	7+m or 3		18		
01111010	disp								
JO = Jump on overflow	<table border="1"><tr><td>01110000</td><td>disp</td></tr></table>	01110000	disp	7+m or 3	7+m or 3		18		
01110000	disp								
JS = Jump on sign	<table border="1"><tr><td>01111000</td><td>disp</td></tr></table>	01111000	disp	7+m or 3	7+m or 3		18		
01111000	disp								
JNE/JNZ = Jump on not equal/not zero	<table border="1"><tr><td>01110101</td><td>disp</td></tr></table>	01110101	disp	7+m or 3	7+m or 3		18		
01110101	disp								
JNL/JGE = Jump on not less/greater or equal	<table border="1"><tr><td>01111101</td><td>disp</td></tr></table>	01111101	disp	7+m or 3	7+m or 3		18		
01111101	disp								
JNLE/JG = Jump on not less or equal/greater	<table border="1"><tr><td>01111111</td><td>disp</td></tr></table>	01111111	disp	7+m or 3	7+m or 3		18		
01111111	disp								
JNB/JAE = Jump on not below/above or equal	<table border="1"><tr><td>01110011</td><td>disp</td></tr></table>	01110011	disp	7+m or 3	7+m or 3		18		
01110011	disp								
JNBE/JA = Jump on not below or equal/above	<table border="1"><tr><td>01110111</td><td>disp</td></tr></table>	01110111	disp	7+m or 3	7+m or 3		18		
01110111	disp								
JNP/JPO = Jump on not par/par odd	<table border="1"><tr><td>01111011</td><td>disp</td></tr></table>	01111011	disp	7+m or 3	7+m or 3		18		
01111011	disp								
JNO = Jump on not overflow	<table border="1"><tr><td>01110001</td><td>disp</td></tr></table>	01110001	disp	7+m or 3	7+m or 3		18		
01110001	disp								
JNS = Jump on not sign	<table border="1"><tr><td>01111001</td><td>disp</td></tr></table>	01111001	disp	7+m or 3	7+m or 3		18		
01111001	disp								
LOOP = Loop CX times	<table border="1"><tr><td>11100010</td><td>disp</td></tr></table>	11100010	disp	8+m or 4	8+m or 4		18		
11100010	disp								
LOOPZ/LOOPE = Loop while zero/equal	<table border="1"><tr><td>11100001</td><td>disp</td></tr></table>	11100001	disp	8+m or 4	8+m or 4		18		
11100001	disp								
LOOPNZ/LOOPNE = Loop while not zero/equal	<table border="1"><tr><td>11100000</td><td>disp</td></tr></table>	11100000	disp	8+m or 4	8+m or 4		18		
11100000	disp								
JCXZ = Jump on CX zero	<table border="1"><tr><td>11100011</td><td>disp</td></tr></table>	11100011	disp	8+m or 4	8+m or 4		18		
11100011	disp								
ENTER = Enter Procedure	<table border="1"><tr><td>11001000</td><td>data-low</td><td>data-high</td><td>L</td></tr></table>	11001000	data-low	data-high	L			2,8	8,9
11001000	data-low	data-high	L						
L = 0		11	11						
L = 1		15	15	2,8	8,9				
L > 1		16+4(L-1)	16+4(L-1)	2,8	8,9				
LEAVE = Leave Procedure	<table border="1"><tr><td>11001001</td></tr></table>	11001001	5	5					
11001001									
INT = Interrupt:									
Type specified	<table border="1"><tr><td>11001101</td><td>type</td></tr></table>	11001101	type	23+m		2,7,8			
11001101	type								
Type 3	<table border="1"><tr><td>11001100</td></tr></table>	11001100	23+m		2,7,8				
11001100									
INTO = Interrupt on overflow	<table border="1"><tr><td>11001110</td></tr></table>	11001110	24+m or 3 (3 if no interrupt)	(3 if no interrupt)	2,6,8				
11001110									

Shaded areas indicate instructions not available in 8086, 88 microsystems.

80C286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS					
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode				
CONTROL TRANSFER (Continued)									
Protected Mode Only:									
Via interrupt or trap gate to same privilege level			40+ m		7,8,11,12,18				
Via interrupt or trap gate to fit different privilege level			78+ m		7,8,11,12,18				
Via Task Gate			167+ m		7,8,11,12,18				
IRET = Interrupt return	<table border="1"><tr><td>11001111</td></tr></table>	11001111	17+ m	31+ m	2,4	8,9,11,12,15,18			
11001111									
Protected Mode Only:									
To different privilege level			55+ m		8,9,11,12,15,18				
To different task (NT = 1)			169+ m		8,9,11,12,18				
BOUND = Detect value out of range	<table border="1"><tr><td>01100010</td><td>mod reg</td><td>r/m</td></tr></table>	01100010	mod reg	r/m	13*	13*	2,6	6,8,9,11,12,18	
01100010	mod reg	r/m							
(Use INT clock count if exception 5)									
PROCESSOR CONTROL									
CLC = Clear carry	<table border="1"><tr><td>11111000</td></tr></table>	11111000	2	2					
11111000									
CMC = Complement carry	<table border="1"><tr><td>11110101</td></tr></table>	11110101	2	2					
11110101									
STC = Set carry	<table border="1"><tr><td>11111001</td></tr></table>	11111001	2	2					
11111001									
CLD = Clear direction	<table border="1"><tr><td>11111100</td></tr></table>	11111100	2	2					
11111100									
STD = Set direction	<table border="1"><tr><td>11111101</td></tr></table>	11111101	2	2					
11111101									
CLI = Clear interrupt	<table border="1"><tr><td>11111010</td></tr></table>	11111010	3	3		14			
11111010									
STI = Set interrupt	<table border="1"><tr><td>11111011</td></tr></table>	11111011	2	2		14			
11111011									
HLT = Halt	<table border="1"><tr><td>11110100</td></tr></table>	11110100	2	2		13			
11110100									
WAIT = Wait	<table border="1"><tr><td>10011011</td></tr></table>	10011011	3	3					
10011011									
LOCK = Bus lock prefix	<table border="1"><tr><td>11110000</td></tr></table>	11110000	0	0		14			
11110000									
CTS = Clear task switched flag	<table border="1"><tr><td>00001111</td><td>00000110</td></tr></table>	00001111	00000110	2	2	3	13		
00001111	00000110								
ESC = Processor Extension Escape	<table border="1"><tr><td>11011TTT</td><td>mod LLL</td><td>r/m</td></tr></table> (TTT LLL are opcode to processor extension)	11011TTT	mod LLL	r/m	9-20*	9-20*	5,8	8,17	
11011TTT	mod LLL	r/m							
SEG = Segment Override Prefix	<table border="1"><tr><td>001 reg 110</td></tr></table>	001 reg 110	0	0					
001 reg 110									
PROTECTION CONTROL									
LGDT = Load global descriptor table register	<table border="1"><tr><td>00001111</td><td>00000001</td><td>mod 010</td><td>r/m</td></tr></table>	00001111	00000001	mod 010	r/m	11*	11*	2,3	9,13
00001111	00000001	mod 010	r/m						
SGDT = Store global descriptor table register	<table border="1"><tr><td>00001111</td><td>00000001</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	00000001	mod 000	r/m	11*	11*	2,3	9
00001111	00000001	mod 000	r/m						
LIDT = Load interrupt descriptor table register	<table border="1"><tr><td>00001111</td><td>00000001</td><td>mod 011</td><td>r/m</td></tr></table>	00001111	00000001	mod 011	r/m	12*	12*	2,3	9,13
00001111	00000001	mod 011	r/m						
SIDT = Store interrupt descriptor table register	<table border="1"><tr><td>00001111</td><td>00000001</td><td>mod 001</td><td>r/m</td></tr></table>	00001111	00000001	mod 001	r/m	12*	12*	2,3	9
00001111	00000001	mod 001	r/m						
LLDT = Load local descriptor table register from register memory	<table border="1"><tr><td>00001111</td><td>00000000</td><td>mod 010</td><td>r/m</td></tr></table>	00001111	00000000	mod 010	r/m		17,19*	1	8,11,13
00001111	00000000	mod 010	r/m						
SLDT = Store local descriptor table register to register/memory	<table border="1"><tr><td>00001111</td><td>00000000</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	00000000	mod 000	r/m		2,3*	1	9
00001111	00000000	mod 000	r/m						

Shaded areas indicate instructions not available in 8086, 88 microsystems.

80C286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS				
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode			
PROTECTION CONTROL (Continued)								
LTR = Local task register from register/memory	<table border="1"><tr><td>00001111</td><td>00000000</td><td>mod 011 r/m</td></tr></table>	00001111	00000000	mod 011 r/m		17,19*	1	9,11,13
00001111	00000000	mod 011 r/m						
STR = Store task register to register memory	<table border="1"><tr><td>00001111</td><td>00000000</td><td>mod 001 r/m</td></tr></table>	00001111	00000000	mod 001 r/m		2,3*	1	9
00001111	00000000	mod 001 r/m						
LMSW = Load machine status word from register/memory	<table border="1"><tr><td>00001111</td><td>00000001</td><td>mod 110 r/m</td></tr></table>	00001111	00000001	mod 110 r/m	3,6*	3,6*	2,3	9,13
00001111	00000001	mod 110 r/m						
SMSW = Store machine status word	<table border="1"><tr><td>00001111</td><td>00000001</td><td>mod 100 r/m</td></tr></table>	00001111	00000001	mod 100 r/m	2,3*	2,3*	2,3	9
00001111	00000001	mod 100 r/m						
LAR = Load access rights from register/memory	<table border="1"><tr><td>00001111</td><td>00000010</td><td>mod reg r/m</td></tr></table>	00001111	00000010	mod reg r/m		14,16*	1	9,11,16
00001111	00000010	mod reg r/m						
LSL = Load segment limit from register/memory	<table border="1"><tr><td>00001111</td><td>00000011</td><td>mod reg r/m</td></tr></table>	00001111	00000011	mod reg r/m		14,16*	1	9,11,16
00001111	00000011	mod reg r/m						
ARPL = Adjust requested privilege level: from register/memory	<table border="1"><tr><td>01100011</td><td>mod reg r/m</td></tr></table>	01100011	mod reg r/m		10*,11*	2	8,9	
01100011	mod reg r/m							
VERR = Verify read access: register/memory	<table border="1"><tr><td>00001111</td><td>00000000</td><td>mod 100 r/m</td></tr></table>	00001111	00000000	mod 100 r/m		14,16*	1	9,11,16
00001111	00000000	mod 100 r/m						
VERR = Verify write access:	<table border="1"><tr><td>00001111</td><td>00000000</td><td>mod 101 r/m</td></tr></table>	00001111	00000000	mod 101 r/m		14,16*	1	9,11,16
00001111	00000000	mod 101 r/m						

Shaded areas indicate instructions not available in 8086, 88 microsystems.

Footnotes

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

- if mod = 11 then r/m is treated as a REG field
- if mod = 00 then DISP = 0*, disp-low and disp-high are absent
- if mod = 01 then DISP = disp-low sign-extended to 16 bits, disp-high is absent
- if mod = 10 then DISP = disp-high: disp-low

- if r/m = 000 then EA = (BX) + (SI) + DISP
- if r/m = 001 then EA = (BX) + (DI) + DISP
- if r/m = 010 then EA = (BP) + (SI) + DISP
- if r/m = 011 then EA = (BP) + (DI) + DISP
- if r/m = 100 then EA = (SI) + DISP
- if r/m = 101 then EA = (DI) + DISP
- if r/m = 110 then EA = (BP) + DISP*
- if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EQ = disp-high: disp-low.

SEGMENT OVERRIDE PREFIX



reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DC

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

DATA SHEET REVISION REVIEW

The following list represents key differences between this and the -001 data sheet. Please review this summary carefully.

1. The typical supply current (I_{CC}) specification has been improved from 180 mA to 125 mA.
2. The "Typical I_{CC} vs Frequency for Different Output Loads" graph has been updated.
3. The package thermal data section has been expanded.
4. The "80C286 Reset Input Timing and Subsequent Processor Cycle Phase" diagram has been updated.
5. The data sheet has been upgraded from ADVANCE INFORMATION to PRELIMINARY.



80286

High Performance Microprocessor with Memory Management and Protection

(80286-12, 80286-10, 80286-8)

- High Performance HMOS III Technology
 - Large Address Space:
 - 16 Megabytes Physical
 - 1 Gigabyte Virtual per Task
 - Integrated Memory Management, Four-Level Memory Protection and Support for Virtual Memory and Operating Systems
 - High Bandwidth Bus Interface (12.5 Megabyte/Sec)
 - Industry Standard O.S. Support:
 - MS-DOS*, UNIX*, XENIX*, iRMX®
 - Two 8086 Upward Compatible Operating Modes:
 - 8086 Real Address Mode
 - Protected Virtual Address Mode
 - Optional Processor Extension:
 - 80287 High Performance 80-bit Numeric Data Processor
 - Range of Clock Rates
 - 12.5 MHz for 80286-12
 - 10 MHz for 80286-10
 - 8 MHz for 80286-8
 - Complete System Development Support:
 - Assembler, PL/M, Pascal, FORTRAN, and In-Circuit-Emulator (ICE™-286)
 - Available in 68 Pin Ceramic LCC (Leadless Chip Carrier), PGA (Pin Grid Array), and PLCC (Plastic Leaded Chip Carrier) Packages
- (See Packaging Spec., Order #231369)

The 80286 is an advanced, high-performance microprocessor with specially optimized capabilities for multiple user and multi-tasking systems. The 80286 has built-in memory protection that supports operating system and task isolation as well as program and data privacy within tasks. A 12 MHz 80286 provides six times or more throughput than the standard 5 MHz 8086. The 80286 includes memory management capabilities that map 2³⁰ (one gigabyte) of virtual address space per task into 2²⁴ bytes (16 megabytes) of physical memory.

The 80286 is upward compatible with 8086 and 88 software. Using 8086 real address mode, the 80286 is object code compatible with existing 8086, 88 software. In protected virtual address mode, the 80286 is source code compatible with 8086, 88 software and may require upgrading to use virtual addresses supported by the 80286's integrated memory management and protection mechanism. Both modes operate at full 80286 performance and execute a superset of the 8086 and 88 instructions.

The 80286 provides special operations to support the efficient implementation and execution of operating systems. For example, one instruction can end execution of one task, save its state, switch to a new task, load its state, and start execution of the new task. The 80286 also supports virtual memory systems by providing a segment-not-present exception and restartable instructions.

*XENIX and MS-DOS are trademarks of Microsoft Corp.

*UNIX is a trademark of Bell Labs or AT&T

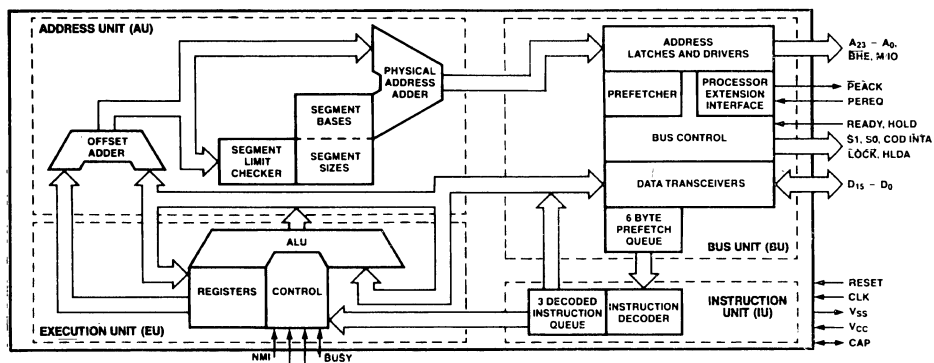
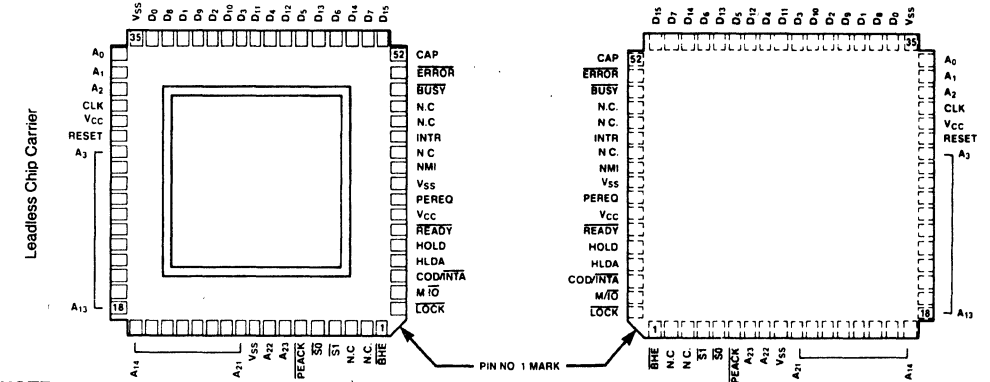
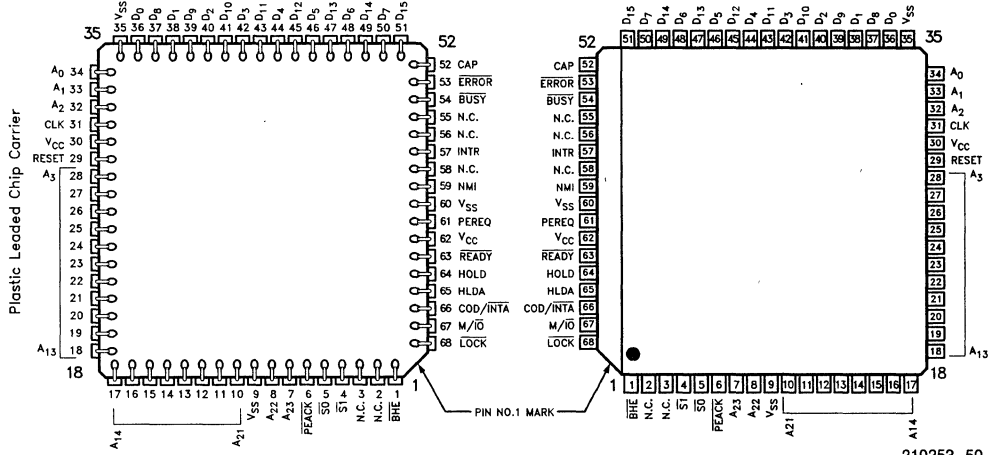


Figure 1. 80286 Internal Block Diagram

210253-1

Component Pad Views—As viewed from underside of component when mounted on the board.

P.C. Board Views—As viewed from the component side of the P.C. board.



NOTE:
N.C. signals must not be connected

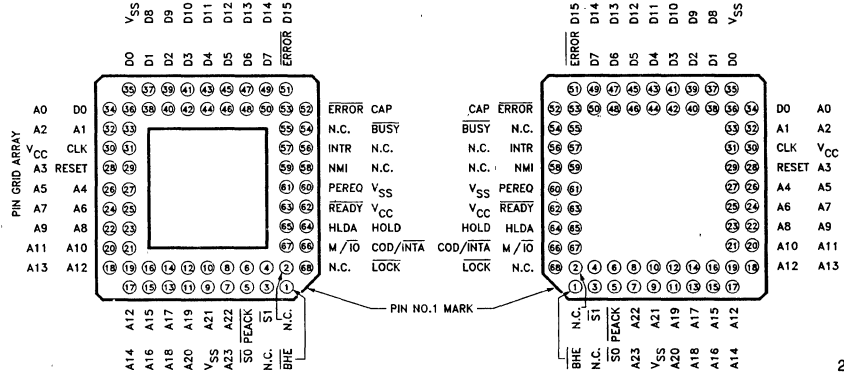


Figure 2. 80286 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for the 80286 microprocessor:

Symbol	Type	Name and Function			
CLK	I	SYSTEM CLOCK provides the fundamental timing for 80286 systems. It is divided by two inside the 80286 to generate the processor clock. The internal divide-by-two circuitry can be synchronized to an external clock generator by a LOW to HIGH transition on the RESET input.			
D ₁₅ -D ₀	I/O	DATA BUS inputs data during memory, I/O, and interrupt acknowledge read cycles; outputs data during memory and I/O write cycles. The data bus is active HIGH and floats to 3-state OFF during bus hold acknowledge.			
A ₂₃ -A ₀	O	ADDRESS BUS outputs physical memory and I/O port addresses. A ₀ is LOW when data is to be transferred on pins D ₇₋₀ . A ₂₃ -A ₁₆ are LOW during I/O transfers. The address bus is active HIGH and floats to 3-state OFF during bus hold acknowledge.			
BHE	O	BUS HIGH ENABLE indicates transfer or data on the upper byte of the data bus. D ₁₅₋₈ . Eight-bit oriented devices assigned to the upper byte of the data bus would normally use BHE to condition chip select functions. BHE is active LOW and floats to 3-state OFF during bus hold acknowledge.			
BHE and A0 Encodings					
	BHE Value	A0 Value			
	Function				
	0	0	Word transfer		
	0	1	Byte transfer on upper half of data bus (D ₁₅ -D ₈)		
	1	0	Byte transfer on lower half of data bus (D ₇ -0)		
	1	1	Will never occur		
S ₁ , S ₀	O	BUS CYCLE STATUS indicates initiation of a bus cycle and, along with M/ \bar{I} O and COD/ \bar{I} NTA, defines the type of bus cycle. The bus is in a T _S state whenever one or both are LOW, S ₁ and S ₀ are active LOW and float to 3-state OFF during bus hold acknowledge.			
80286 Bus Cycle Status Definition					
	COD/\bar{I}NTA	M/\bar{I}O	S₁	S₀	Bus Cycle Initiated
	0 (LOW)	0	0	0	Interrupt acknowledge
	0	0	0	1	Will not occur
	0	0	1	0	Will not occur
	0	0	1	1	None; not a status cycle
	0	1	0	0	IF A1 = 1 then halt; else shutdown
	0	1	0	1	Memory data read
	0	1	1	0	Memory data write
	0	1	1	1	None; not a status cycle
	1 (HIGH)	0	0	0	Will not occur
	1	0	0	1	I/O read
	1	0	1	0	I/O write
	1	0	1	1	None; not a status cycle
	1	1	0	0	Will not occur
	1	1	0	1	Memory instruction read
	1	1	1	0	Will not occur
	1	1	1	1	None; not a status cycle
M/ \bar{I} O	O	MEMORY I/O SELECT distinguishes memory access from I/O access. If HIGH during T _S , a memory cycle or a halt/shutdown cycle is in progress. If LOW, an I/O cycle or an interrupt acknowledge cycle is in progress. M/ \bar{I} O floats to 3-state OFF during bus hold acknowledge.			
COD/ \bar{I} NTA	O	CODE/INTERRUPT ACKNOWLEDGE distinguishes instruction fetch cycles from memory data read cycles. Also distinguishes interrupt acknowledge cycles from I/O cycles. COD/ \bar{I} NTA floats to 3-state OFF during bus hold acknowledge. Its timing is the same as M/ \bar{I} O.			
LOCK	O	BUS LOCK indicates that other system bus masters are not to gain control of the system bus for the current and the following bus cycle. The LOCK signal may be activated explicitly by the "LOCK" instruction prefix or automatically by 80286 hardware during memory XCHG instructions, interrupt acknowledge, or descriptor table access. LOCK is active LOW and floats to 3-state OFF during bus hold acknowledge.			
READY	I	BUS READY terminates a bus cycle. Bus cycles are extended without limit until terminated by READY LOW. READY is an active LOW synchronous input requiring setup and hold times relative to the system clock be met for correct operation. READY is ignored during bus hold acknowledge.			

Table 1. Pin Description (Continued)

Symbol	Type	Name and Function
HOLD HLDA	I O	BUS HOLD REQUEST AND HOLD ACKNOWLEDGE control ownership of the 80286 local bus. The HOLD input allows another local bus master to request control of the local bus. When control is granted, the 80286 will float its bus drivers to 3-state OFF and then activate HLDA, thus entering the bus hold acknowledge condition. The local bus will remain granted to the requesting master until HOLD becomes inactive which results in the 80286 deactivating HLDA and regaining control of the local bus. This terminates the bus hold acknowledge condition. HOLD may be asynchronous to the system clock. These signals are active HIGH.
INTR	I	INTERRUPT REQUEST requests the 80286 to suspend its current program execution and service a pending external request. Interrupt requests are masked whenever the interrupt enable bit in the flag word is cleared. When the 80286 responds to an interrupt request, it performs two interrupt acknowledge bus cycles to read an 8-bit interrupt vector that identifies the source of the interrupt. To assure program interruption, INTR must remain active until the first interrupt acknowledge cycle is completed. INTR is sampled at the beginning of each processor cycle and must be active HIGH at least two processor cycles before the current instruction ends in order to interrupt before the next instruction. INTR is level sensitive, active HIGH, and may be asynchronous to the system clock.
NMI	I	NON-MASKABLE INTERRUPT REQUEST interrupts the 80286 with an internally supplied vector value of 2. No interrupt acknowledge cycles are performed. The interrupt enable bit in the 80286 flag word does not affect this input. The NMI input is active HIGH, may be asynchronous to the system clock, and is edge triggered after internal synchronization. For proper recognition, the input must have been previously LOW for at least four system clock cycles and remain HIGH for at least four system clock cycles.
PEREQ PEACK	I O	PROCESSOR EXTENSION OPERAND REQUEST AND ACKNOWLEDGE extend the memory management and protection capabilities of the 80286 to processor extensions. The PEREQ input requests the 80286 to perform a data operand transfer for a processor extension. The PEACK output signals the processor extension when the requested operand is being transferred. PEREQ is active HIGH and floats to 3-state OFF during bus hold acknowledge. PEACK may be asynchronous to the system clock. PEACK is active LOW.
BUSY ERROR	I I	PROCESSOR EXTENSION BUSY AND ERROR indicate the operating condition of a processor extension to the 80286. An active BUSY input stops 80286 program execution on WAIT and some ESC instructions until BUSY becomes inactive (HIGH). The 80286 may be interrupted while waiting for BUSY to become inactive. An active ERROR input causes the 80286 to perform a processor extension interrupt when executing WAIT or some ESC instructions. These inputs are active LOW and may be asynchronous to the system clock. These inputs have internal pull-up resistors.

Table 1. Pin Description (Continued)

Symbol	Type	Name and Function	
RESET	I	<p>SYSTEM RESET clears the internal logic of the 80286 and is active HIGH. The 80286 may be reinitialized at any time with a LOW to HIGH transition on RESET which remains active for more than 16 system clock cycles. During RESET active, the output pins of the 80286 enter the state shown below:</p>	
		80286 Pin State During Reset	
		Pin Value	Pin Names
		1 (HIGH) 0 (LOW) 3-state OFF	S0, S1, PEACK, A23-A0, BHE, LOCK M/I0, COD/INTA, HLDA (Note 1) D15-D0
		<p>Operation of the 80286 begins after a HIGH to LOW transition on RESET. The HIGH to LOW transition of RESET must be synchronous to the system clock. Approximately 38 CLK cycles from the trailing edge of RESET are required by the 80286 for internal initialization before the first bus cycle, to fetch code from the power-on execution address, occurs.</p> <p>A LOW to HIGH transition of RESET synchronous to the system clock will end a processor cycle at the second HIGH to LOW transition of the system clock. The LOW to HIGH transition of RESET may be asynchronous to the system clock; however, in this case it cannot be predetermined which phase of the processor clock will occur during the next system clock period. Synchronous LOW to HIGH transitions of RESET are required only for systems where the processor clock must be phase synchronous to another clock.</p>	
V _{SS}	I	SYSTEM GROUND: 0 Volts.	
V _{CC}	I	SYSTEM POWER: + 5 Volt Power Supply.	
CAP	I	<p>SUBSTRATE FILTER CAPACITOR: a 0.047 μF \pm 20% 12V capacitor must be connected between this pin and ground. This capacitor filters the output of the internal substrate bias generator. A maximum DC leakage current of 1 μA is allowed through the capacitor.</p> <p>For correct operation of the 80286, the substrate bias generator must charge this capacitor to its operating voltage. The capacitor chargeup time is 5 milliseconds (max.) after V_{CC} and CLK reach their specified AC and DC parameters. RESET may be applied to prevent spurious activity by the CPU during this time. After this time, the 80286 processor clock can be synchronized to another clock by pulsing RESET LOW synchronous to the system clock.</p>	

NOTE:

1. HLDA is only Low if HOLD is inactive (Low).

FUNCTIONAL DESCRIPTION

Introduction

The 80286 is an advanced, high-performance micro-processor with specially optimized capabilities for multiple user and multi-tasking systems. Depending on the application, a 12 MHz 80286's performance is up to six times faster than the standard 5 MHz 8086's, while providing complete upward software compatibility with Intel's 8086, 88, and 186 family of CPU's.

The 80286 operates in two modes: 8086 real address mode and protected virtual address mode. Both modes execute a superset of the 8086 and 88 instruction set.

In 8086 real address mode programs use real addresses with up to one megabyte of address space. Programs use virtual addresses in protected virtual address mode, also called protected mode. In protected mode, the 80286 CPU automatically maps 1 gigabyte of virtual addresses per task into a 16 megabyte real address space. This mode also provides memory protection to isolate the operating system and ensure privacy of each tasks' programs and data. Both modes provide the same base instruction set, registers, and addressing modes.

The following Functional Description describes first, the base 80286 architecture common to both modes, second, 8086 real address mode, and third, protected mode.

80286 BASE ARCHITECTURE

The 8086, 88, 186, and 286 CPU family all contain the same basic set of registers, instructions, and

addressing modes. The 80286 processor is upward compatible with the 8086, 8088, and 80186 CPU's.

Register Set

The 80286 base architecture has fifteen registers as shown in Figure 3. These registers are grouped into the following four categories:

General Registers: Eight 16-bit general purpose registers used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used either in their entirety as 16-bit words or split into pairs of separate 8-bit registers.

Segment Registers: Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers: Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode determines the specific registers used for operand address calculations.

Status and Control Registers: The 3 16-bit special purpose registers in figure 3A record or control certain aspects of the 80286 processor state including the Instruction Pointer, which contains the offset address of the next sequential instruction to be executed.

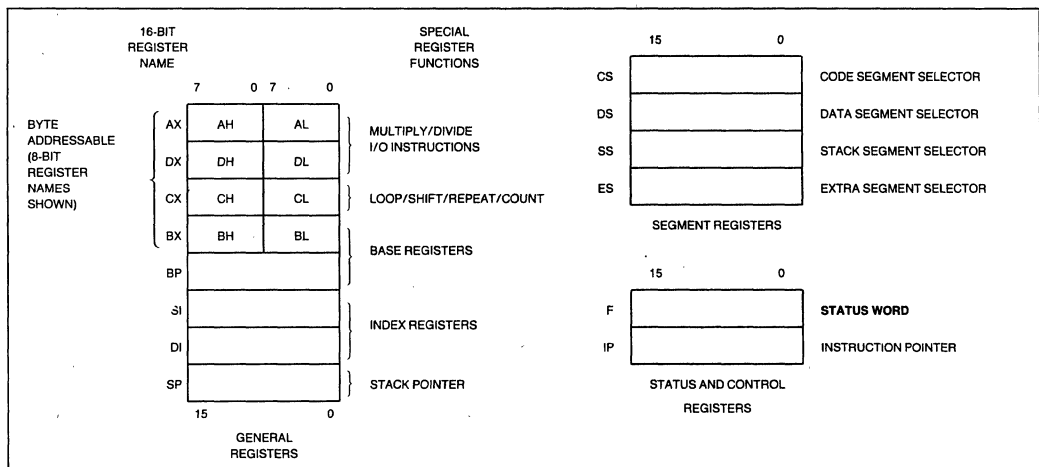


Figure 3. Register Set

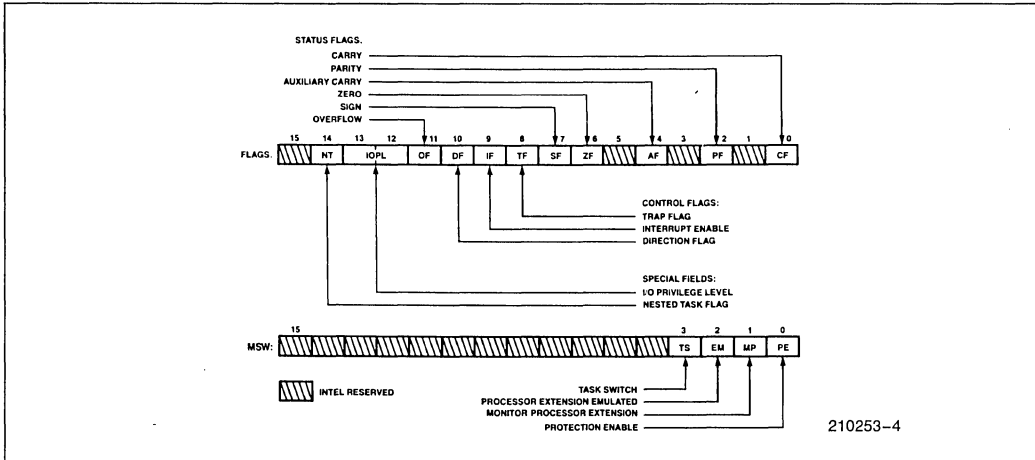


Figure 3a. Status and Control Register Bit Functions

Flags Word Description

The Flags word (Flags) records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80286 within a given operating mode (bits 8 and 9). Flags is a 16-bit register. The function of the flag bits is given in Table 2.

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high level instructions, and processor control. These categories are summarized in Figure 4.

An 80286 instruction can reference zero, one, or two operands; where an operand resides in a register, in the instruction itself, or in memory. Zero-operand instructions (e.g. NOP and HLT) are usually one byte long. One-operand instructions (e.g. INC and DEC) are usually two bytes long but some are encoded in only one byte. One-operand instructions may reference a register or memory location. Two-operand instructions permit the following six types of instruction operations:

- Register to Register
- Memory to Register
- Immediate to Register
- Memory to Memory
- Register to Memory
- Immediate to Memory

Table 2. Flags Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag—Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
11	OF	Overflow Flag—Set if result is a too-large positive number or a too-small negative number (excluding sign-bit) to fit in destination operand; cleared otherwise
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index registers when set. Clearing DF causes auto increment.

Two-operand instructions (e.g. MOV and ADD) are usually three to six bytes long. Memory to memory operations are provided by a special class of string instructions requiring one to three bytes. For detailed instruction formats and encodings refer to the instruction set summary at the end of this document.

For detailed operation and usage of each instruction, see Appendix of 80286 Programmer's Reference Manual (Order No. 210498)

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack

Figure 4a. Data Transfer Instructions

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero

Figure 4c. String Instructions

ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiple byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

Figure 4b. Arithmetic Instructions

LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

Figure 4d. Shift/Rotate Logical Instructions

CONDITIONAL TRANSFERS		UNCONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal	CALL	Call procedure
JAE/JNB	Jump if above or equal/not below	RET	Return from procedure
JB/JNAE	Jump if below/not above nor equal	JMP	Jump
JBE/JNA	Jump if below or equal/not above		
JC	Jump if carry	ITERATION CONTROLS	
JE/JZ	Jump if equal/zero	LOOP	Loop
JG/JNLE	Jump if greater/not less nor equal		
JGE/JNL	Jump if greater or equal/not less	LOOPE/LOOPZ	Loop if equal/zero
JL/JNGE	Jump if less/not greater nor equal	LOOPNE/LOOPNZ	Loop if not equal/not zero
JLE/JNG	Jump if less or equal/not greater	JCXZ	Jump if register CX = 0
JNC	Jump if not carry	INTERRUPTS	
JNE/JNZ	Jump if not equal/not zero	INT	Interrupt
JNO	Jump if not overflow		
JNP/JPO	Jump if not parity/parity odd	INTO	Interrupt if overflow
JNS	Jump if not sign	IRET	Interrupt return
JO	Jump if overflow		
JP/JPE	Jump if parity/parity even		
JS	Jump if sign		

Figure 4e. Program Transfer Instructions

FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for BUSY not active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
EXECUTION ENVIRONMENT CONTROL	
LMSW	Load machine status word
SMSW	Store machine status word

Figure 4f. Processor Control Instructions

ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4g. High Level Instructions

Memory Organization

Memory is organized as sets of variable length segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two component address (a pointer) that consists of a 16-bit segment selector, and a 16-bit offset. The segment selector indicates the desired segment in memory. The offset component indicates the desired byte address within the segment.

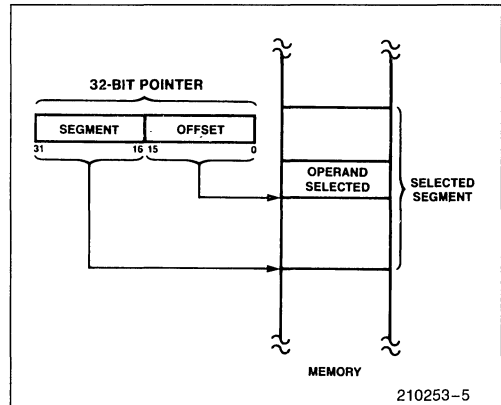


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Automatic with instruction prefetch
Stack	Stack (SS)	All stack pushes and pops. Any memory reference which uses BP as a base register.
Local Data	Data (DS)	All data references except when relative to stack or string destination
External (Global) Data	Extra (ES)	Alternate data segment and destination of string operation

All instructions that address operands in memory must specify the segment and the offset. For speed and compact instruction encoding, segment selectors are usually stored in the high speed segment registers. An instruction need specify only the desired segment register and an offset in order to address a memory operand.

Most instructions need not explicitly specify which segment register is used. The correct segment register is automatically chosen according to the rules of Table 3. These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs. To access operands not residing in one of the four immediately available segments, a full 32-bit pointer or a new segment selector must be loaded.

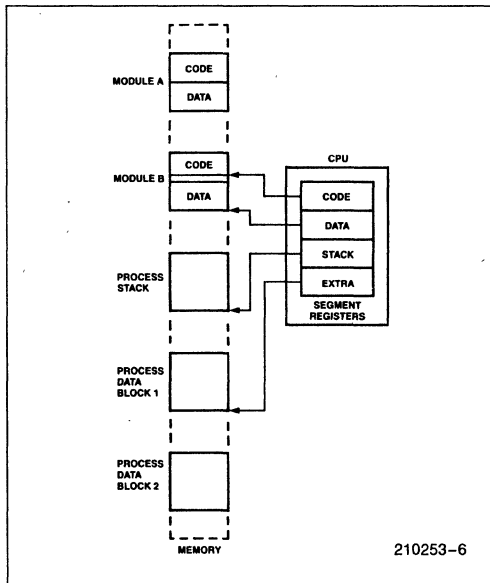


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80286 provides a total of eight addressing modes for instructions to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8 or 16-bit general registers.

Immediate Operand Mode: The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: segment selector and offset. The segment selector is supplied by a segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset is calculated by summing any combination of the following three address elements:

the **displacement** (an 8 or 16-bit immediate value contained in the instruction)

the **base** (contents of either the BX or BP base registers)

the **index** (contents of either the SI or DI index registers)

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

Direct Mode: The operand's offset is contained in the instruction as an 8 or 16-bit displacement element.

Register Indirect Mode: The operand's offset is in one of the registers SI, DI, BX, or BP.

Based Mode: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of a base register (BX or BP).

Indexed Mode: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of an index register (SI or DI).

Based Indexed Mode: The operand's offset is the sum of the contents of a base register and an index register.

Based Indexed Mode with Displacement: The operand's offset is the sum of a base register's contents, an index register's contents, and an 8 or 16-bit displacement.

Data Types

The 80286 directly supports the following data types:

- Integer:** A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32 and 64-bit integers are supported using the Numeric Data Processor, the 80287.
- Ordinal:** An unsigned binary numeric value contained in an 8-bit byte or 16-bit word.
- Pointer:** A 32-bit quantity, composed of a segment selector component and an offset component. Each component is a 16-bit word.
- String:** A contiguous sequence of bytes or words. A string may contain from 1 byte to 64K bytes.
- ASCII:** A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- BCD:** A byte (unpacked) representation of the decimal digits 0–9.
- Packed BCD:** A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble of the byte.
- Floating Point:** A signed 32, 64, or 80-bit real number representation. (Floating point operands are supported using the 80287 Numeric Processor).

Figure 7 graphically represents the data types supported by the 80286.

either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A₁₅–A₈ are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

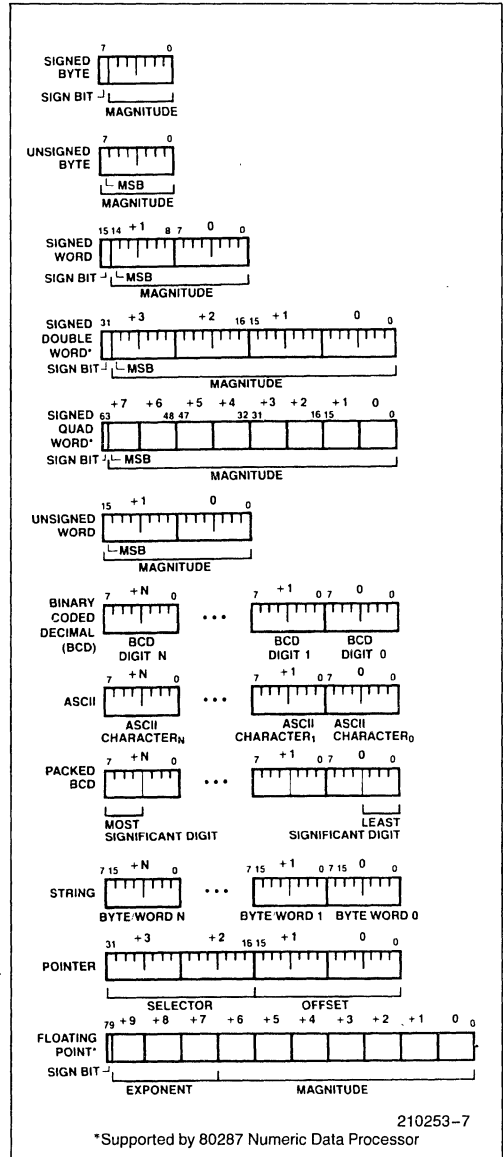


Figure 7. 80286 Supported Data Types

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. I/O instructions address the I/O space with

Table 4. Interrupt Vector Assignments

Function	Interrupt Number	Related Instructions	Does Return Address Point to Instruction Causing Exception?
Divide error exception	0	DIV, IDIV	Yes
Single step interrupt	1	All	
NMI interrupt	2	INT 2 or NMI pin	
Breakpoint interrupt	3	INT 3	
INTO detected overflow exception	4	INTO	No
BOUND range exceeded exception	5	BOUND	Yes
Invalid opcode exception	6	Any undefined opcode	Yes
Processor extension not available exception	7	ESC or WAIT	Yes
Intel reserved—do not use	8-15		
Processor extension error interrupt	16	ESC or WAIT	
Intel reserved—do not use	17-31		
User defined	32-255		

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Flags) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable. Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. The return address from an exception will always point at the instruction causing the exception and include any leading instruction prefixes.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. For each interrupt, an 8-bit vector must be supplied to the 80286 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

MASKABLE INTERRUPT (INTR)

The 80286 provides a maskable hardware interrupt request pin, INTR. Software enables this input by

setting the interrupt flag bit (IF) in the flag word. All 224 user-defined interrupt sources can share this input, yet they can retain separate interrupt handlers. An 8-bit vector read by the CPU during the interrupt acknowledge sequence (discussed in System Interface section) identifies the source of the interrupt.

Further maskable interrupts are disabled while servicing an interrupt by resetting the IF bit as part of the response to an interrupt or exception. The saved flag word will reflect the enable status of the processor prior to the interrupt. Until the flag word is restored to the flag register, the interrupt flag will be zero unless specifically set. The interrupt return instruction includes restoring the flag word, thereby restoring the original status of IF.

NON-MASKABLE INTERRUPT REQUEST (NMI)

A non-maskable interrupt input (NMI) is also provided. NMI has higher priority than INTR. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed.

While executing the NMI servicing procedure, the 80286 will service neither further NMI requests, INTR requests, nor the processor extension segment overrun interrupt until an interrupt return (IRET) instruction is executed or the CPU is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. IF is cleared at the beginning of an NMI interrupt to inhibit INTR interrupts.

SINGLE STEP INTERRUPT

The 80286 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single step interrupt and is controlled by the single step flag bit (TF) in the flag word. Once this bit is set, an internal single step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single stepped.

Interrupt Priorities

When simultaneous interrupt requests occur, they are processed in a fixed order as shown in Table 5. Interrupt processing involves saving the flags, return address, and setting CS:IP to point at the first instruction of the interrupt handler. If other interrupts remain enabled they are processed before the first instruction of the current interrupt handler is executed. The last interrupt processed is therefore the first one serviced.

Table 5. Interrupt Processing Order

Order	Interrupt
1	Instruction exception
2	Single step
3	NMI
4	Processor extension segment overrun
5	INTR
6	INT instruction

Initialization and Processor Reset

Processor initialization or start up is accomplished by driving the RESET input pin HIGH. RESET forces the 80286 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active. After RESET becomes inactive and an internal processing interval elapses, the 80286 begins execution in real address mode with the instruction at physical location FFFF0(H). RESET also sets some registers to predefined values as shown in Table 6.

Table 6. 80286 Initial Register State after RESET

Flag word	0002(H)
Machine Status Word	FFF0(H)
Instruction pointer	FFF0(H)
Code segment	F000(H)
Data segment	0000(H)
Extra segment	0000(H)
Stack segment	0000(H)

HOLD must not be active during the time from the leading edge of RESET to 34 CLKs after the trailing edge of RESET.

Machine Status Word Description

The machine status word (MSW) records when a task switch takes place and controls the operating mode of the 80286. It is a 16-bit register of which the lower four bits are used. One bit places the CPU into protected mode, while the other three bits, as shown in Table 7, control the processor extension interface. After RESET, this register contains FFF0(H) which places the 80286 in 8086 real address mode.

Table 7. MSW Bit Functions

Bit Position	Name	Function
0	PE	Protected mode enable places the 80286 into protected mode and cannot be cleared except by RESET.
1	MP	Monitor processor extension allows WAIT instructions to cause a processor extension not present exception (number 7).
2	EM	Emulate processor extension causes a processor extension not present exception (number 7) on ESC instructions to allow emulating a processor extension.
3	TS	Task switched indicates the next instruction using a processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task.

The LMSW and SMSW instructions can load and store the MSW in real address mode. The recommended use of TS, EM, and MP is shown in Table 8.

Table 8. Recommended MSW Encodings For Processor Extension Control

TS	MP	EM	Recommended Use	Instructions Causing Exception 7
0	0	0	Initial encoding after RESET. 80286 operation is identical to 8086, 88.	None
0	0	1	No processor extension is available. Software will emulate its function.	ESC
1	0	1	No processor extension is available. Software will emulate its function. The current processor extension context may belong to another task.	ESC
0	1	0	A processor extension exists.	None
1	1	0	A processor extension exists. The current processor extension context may belong to another task. The Exception 7 on WAIT allows software to test for an error pending from a previous processor extension operation.	ESC or WAIT

Halt

The HLT instruction stops program execution and prevents the CPU from using the local bus until restarted. Either NMI, INTR with IF = 1, or RESET will force the 80286 out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

8086 REAL ADDRESS MODE

The 80286 executes a fully upward-compatible superset of the 8086 instruction set in real address mode. In real address mode the 80286 is object code compatible with 8086 and 8088 software. The real address mode architecture (registers and addressing modes) is exactly as described in the 80286 Base Architecture section of this Functional Description.

Memory Size

Physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pins A₀ through A₁₉ and BHE. Address bits A₂₀ through A₂₃ should be ignored.

Memory Addressing

In real address mode physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pins A₀ through A₁₉ and BHE. Address bits A₂₀–A₂₃ may not always be zero in real mode. A₂₀–A₂₃ should not be used by the system while the 80286 is operating in Real Mode.

The selector portion of a pointer is interpreted as the upper 16 bits of a 20-bit segment address. The lower four bits of the 20-bit segment address are always zero. Segment addresses, therefore, begin on multiples of 16 bytes. See Figure 8 for a graphic representation of address information.

All segments in real address mode are 64K bytes in size and may be read, written, or executed. An exception or interrupt can occur if data operands or instructions attempt to wrap around the end of a segment (e.g. a word with its low order byte at offset FFFF(H) and its high order byte at offset 0000(H)). If, in real address mode, the information contained in a segment does not use the full 64K bytes, the unused end of the segment may be overlaid by another segment to reduce physical memory requirements.

Reserved Memory Locations

The 80286 reserves two fixed areas of memory in real address mode (see Figure 9); system initializa-

tion area and interrupt table area. Locations from addresses FFFF0(H) through FFFFF(H) are reserved for system initialization. Initial execution begins at location FFFF0(H). Locations 00000(H) through 003FF(H) are reserved for interrupt vectors.

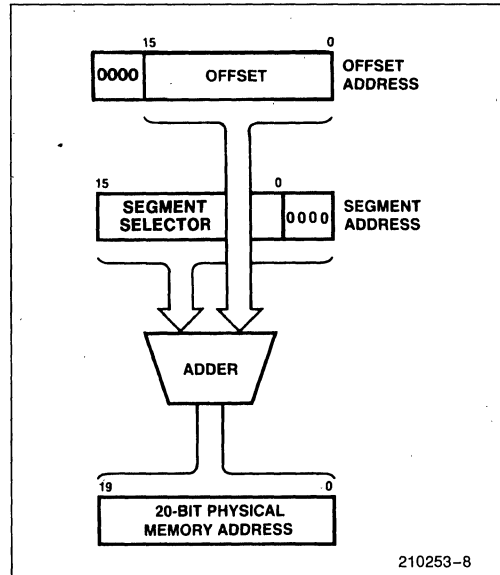


Figure 8. 8086 Real Address Mode Address Calculation

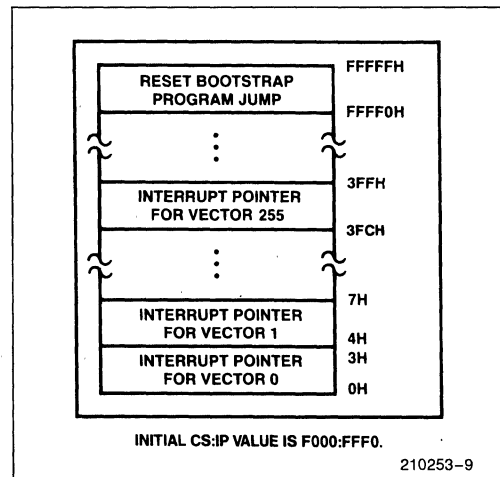


Figure 9. 8086 Real Address Mode Initially Reserved Memory Locations

Table 9. Real Address Mode Addressing Interrupts

Function	Interrupt Number	Related Instructions	Return Address Before Instruction?
Interrupt table limit too small exception	8	INT vector is not within table limit	Yes
Processor extension segment overrun interrupt	9	ESC with memory operand extending beyond offset FFFF(H)	No
Segment overrun exception	13	Word memory reference with offset = FFFF(H) or an attempt to execute past the end of a segment	Yes

Interrupts

Table 9 shows the interrupt vectors reserved for exceptions and interrupts which indicate an addressing error. The exceptions leave the CPU in the state existing before attempting to execute the failing instruction (except for PUSH, POP, PUSH, or POPA). Refer to the next section on protected mode initialization for a discussion on exception 8.

Protected Mode Initialization

To prepare the 80286 for protected mode, the LIDT instruction is used to load the 24-bit interrupt table base and 16-bit limit for the protected mode interrupt table. This instruction can also set a base and limit for the interrupt vector table in real address mode. After reset, the interrupt table base is initialized to 000000(H) and its size set to 03FF(H). These values are compatible with 8086, 88 software. LIDT should only be executed in preparation for protected mode.

Shutdown

Shutdown occurs when a severe error is detected that prevents further instruction processing by the CPU. Shutdown and halt are externally signalled via a halt bus operation. They can be distinguished by A_1 HIGH for halt and A_1 LOW for shutdown. In real address mode, shutdown can occur under two conditions:

- Exceptions 8 or 13 happen and the IDT limit does not include the interrupt vector.
- A CALL INT or PUSH instruction attempts to wrap around the stack segment when SP is not even.

An NMI input can bring the CPU out of shutdown if the IDT limit is at least 000F(H) and SP is greater than 0005(H), otherwise shutdown can only be exited via the RESET input.

PROTECTED VIRTUAL ADDRESS MODE

The 80286 executes a fully upward-compatible superset of the 8086 instruction set in protected virtual address mode (protected mode). Protected mode also provides memory management and protection mechanisms and associated instructions.

The 80286 enters protected virtual address mode from real address mode by setting the PE (Protection Enable) bit of the machine status word with the Load Machine Status Word (LMSW) instruction. Protected mode offers extended physical and virtual memory address space, memory protection mechanisms, and new operations to support operating systems and virtual memory.

All registers, instructions, and addressing modes described in the 80286 Base Architecture section of this Functional Description remain the same. Programs for the 8086, 88, 186, and real address mode 80286 can be run in protected mode; however, embedded constants for segment selectors are different.

Memory Size

The protected mode 80286 provides a 1 gigabyte virtual address space per task mapped into a 16 megabyte physical address space defined by the address pin A_{23-A_0} and \overline{BHE} . The virtual address space may be larger than the physical address space since any use of an address that does not map to a physical memory location will cause a restartable exception.

Memory Addressing

As in real address mode, protected mode uses 32-bit pointers, consisting of 16-bit selector and offset components. The selector, however, specifies an index into a memory resident table rather than the upper 16-bits of a real memory address. The 24-bit

base address of the desired segment is obtained from the tables in memory. The 16-bit offset is added to the segment base address to form the physical address as shown in Figure 10. The tables are automatically referenced by the CPU whenever a segment register is loaded with a selector. All 80286 instructions which load a segment register will reference the memory based tables without additional software. The memory based tables contain 8 byte values called descriptors.

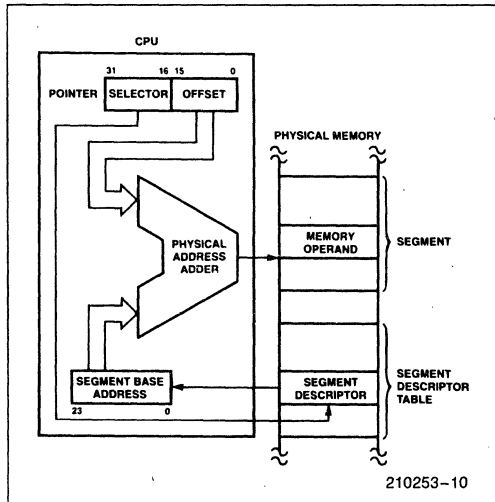


Figure 10. Protected Mode Memory Addressing

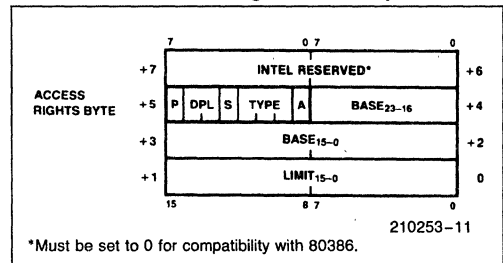
DESCRIPTORS

Descriptors define the use of memory. Special types of descriptors also define new functions for transfer of control and task switching. The 80286 has segment descriptors for code, stack and data segments, and system control descriptors for special system data segments and control transfer operations. Descriptor accesses are performed as locked bus operations to assure descriptor integrity in multi-processor systems.

CODE AND DATA SEGMENT DESCRIPTORS (S = 1)

Besides segment base addresses, code and data descriptors contain other segment attributes including segment size (1 to 64K bytes), access rights (read only, read/write, execute only, and execute/read), and presence in memory (for virtual memory systems) (See Figure 11). Any segment usage violating a segment attribute indicated by the segment descriptor will prevent the memory cycle and cause an exception or interrupt.

Code or Data Segment Descriptor



*Must be set to 0 for compatibility with 80386.

Access Rights Byte Definition

Bit Position	Name	Function
7	Present (P)	P = 1 Segment is mapped into physical memory. P = 0 No mapping to physical memory exists, base and limit are not used.
6-5	Descriptor Privilege Level (DPL)	Segment privilege attribute used in privilege tests.
4	Segment Descriptor (S)	S = 1 Code or Data (includes stacks) segment descriptor S = 0 System Segment Descriptor or Gate Descriptor
3	Executable (E) Expansion Direction (ED)	E = 0 Data segment descriptor type is:
2		ED = 0 Expand up segment, offsets must be ≤ limit. ED = 1 Expand down segment, offsets must be > limit.
1	Writeable (W)	W = 0 Data segment may not be written into.
		W = 1 Data segment may be written into.
3	Executable (E) Conforming (C)	E = 1 Code Segment Descriptor type is:
2		C = 1 Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.
1	Readable (R)	R = 0 Code segment may not be read
		R = 1 Code segment may be read.
0	Accessed (A)	A = 0 Segment has not been accessed. A = 1 Segment selector has been loaded into segment register or used by selector test instructions.

Type Field Definition

Figure 11. Code and Data Segment Descriptor Formats

Code and data (including stack data) are stored in two types of segments: code segments and data segments. Both types are identified and defined by segment descriptors ($S = 1$). Code segments are identified by the executable (E) bit set to 1 in the descriptor access rights byte. The access rights byte of both code and data segment descriptor types have three fields in common: present (P) bit, Descriptor Privilege Level (DPL), and accessed (A) bit. If $P = 0$, any attempted use of this segment will cause a not-present exception. DPL specifies the privilege level of the segment descriptor. DPL controls when the descriptor may be used by a task (refer to privilege discussion below). The A bit shows whether the segment has been previously accessed for usage profiling, a necessity for virtual memory systems. The CPU will always set this bit when accessing the descriptor.

Data segments ($S = 1, E = 0$) may be either read-only or read-write as controlled by the W bit of the access rights byte. Read-only ($W = 0$) data segments may not be written into. Data segments may grow in two directions, as determined by the Expansion Direction (ED) bit: upwards ($ED = 0$) for data segments, and downwards ($ED = 1$) for a segment containing a stack. The limit field for a data segment descriptor is interpreted differently depending on the ED bit (see Figure 11).

A code segment ($S = 1, E = 1$) may be execute-only or execute/read as determined by the Readable (R) bit. Code segments may never be written into and execute-only code segments ($R = 0$) may not be read. A code segment may also have an attribute called conforming (C). A conforming code segment may be shared by programs that execute at different privilege levels. The DPL of a conforming code segment defines the range of privilege levels at which the segment may be executed (refer to privilege discussion below). The limit field identifies the last byte of a code segment.

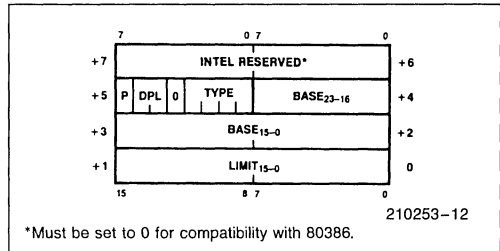
SYSTEM SEGMENT DESCRIPTORS ($S = 0, TYPE = 1-3$)

In addition to code and data segment descriptors, the protected mode 80286 defines System Segment Descriptors. These descriptors define special system data segments which contain a table of descriptors (Local Descriptor Table Descriptor) or segments which contain the execution state of a task (Task State Segment Descriptor).

Figure 12 gives the formats for the special system data segment descriptors. The descriptors contain a 24-bit base address of the segment and a 16-bit limit. The access byte defines the type of descriptor, its state and privilege level. The descriptor contents are valid and the segment is in physical memory if $P = 1$. If $P = 0$, the segment is not valid. The DPL field is only used in Task State Segment descriptors and indicates the privilege level at which the descrip-

tor may be used (see Privilege). Since the Local Descriptor Table descriptor may only be used by a special privileged instruction, the DPL field is not used. Bit 4 of the access byte is 0 to indicate that it is a system control descriptor. The type field specifies the descriptor type as indicated in Figure 12.

System Segment Descriptor



System Segment Descriptor Fields

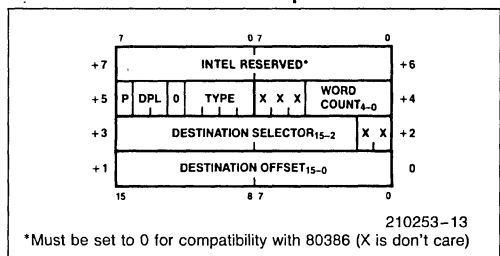
Name	Value	Description
TYPE	1	Available Task State Segment (TSS)
	2	Local Descriptor Table
	3	Busy Task State Segment (TSS)
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid
DPL	0-3	Descriptor Privilege Level
BASE	24-bit number	Base Address of special system data segment in real memory
LIMIT	16-bit number	Offset of last byte in segment

Figure 12. System Segment Descriptor Format

GATE DESCRIPTORS ($S = 0, TYPE = 4-7$)

Gates are used to control access to entry points within the target code segment. The gate descriptors are call gates, task gates, interrupt gates and trap gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the CPU to automatically perform protection checks and control entry point of the destination. Call gates are used to change privilege levels (see Privilege), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines. The interrupt gate disables interrupts (resets IF) while the trap gate does not.

Gate Descriptor



Gate Descriptor Fields

Name	Value	Description
TYPE	4	-Call Gate
	5	-Task Gate
	6	-Interrupt Gate
	7	-Trap Gate
P	0	-Descriptor Contents are not valid
	1	-Descriptor Contents are valid
DPL	0-3	Descriptor Privilege Level
WORD COUNT	0-31	Number of words to copy from callers stack to called procedures stack. Only used with call gate.
DESTINATION SELECTOR	16-bit selector	Selector to the target code segment (Call, Interrupt or Trap Gate)
		Selector to the target task state segment (Task Gate)
DESTINATION OFFSET	16-bit offset	Entry point within the target code segment

Figure 13. Gate Descriptor Format

Figure 13 shows the format of the gate descriptors. The descriptor contains a destination pointer that points to the descriptor of the target segment and the entry point offset. The destination selector in an interrupt gate, trap gate, and call gate must refer to a code segment descriptor. These gate descriptors contain the entry point to prevent a program from constructing and using an illegal entry point. Task gates may only refer to a task state segment. Since task gates invoke a task switch, the destination offset is not used in the task gate.

Exception 13 is generated when the gate is used if a destination selector does not refer to the correct descriptor type. The word count field is used in the call gate descriptor to indicate the number of parameters (0-31 words) to be automatically copied from the caller's stack to the stack of the called routine when a control transfer changes privilege levels. The word count field is not used by any other gate descriptor.

The access byte format is the same for all gate descriptors. P = 1 indicates that the gate contents are valid. P = 0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the de-

scriptor privilege level and specifies when this descriptor may be used by a task (refer to privilege discussion below). Bit 4 must equal 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 13.

SEGMENT DESCRIPTOR CACHE REGISTERS

A segment descriptor cache register is assigned to each of the four segment registers (CS, SS, DS, ES). Segment descriptors are automatically loaded (cached) into a segment descriptor cache register (Figure 14) whenever the associated segment register is loaded with a selector. Only segment descriptors may be loaded into segment descriptor cache registers. Once loaded, all references to that segment of memory use the cached descriptor information instead of reaccessing the descriptor. The descriptor cache registers are not visible to programs. No instructions exist to store their contents. They only change when a segment register is loaded.

SELECTOR FIELDS

A protected mode selector has three fields: descriptor entry index, local or global descriptor table indicator (TI), and selector privilege (RPL) as shown in Figure 15. These fields select one of two memory based tables of descriptors, select the appropriate table entry and allow highspeed testing of the selector's privilege attribute (refer to privilege discussion below).

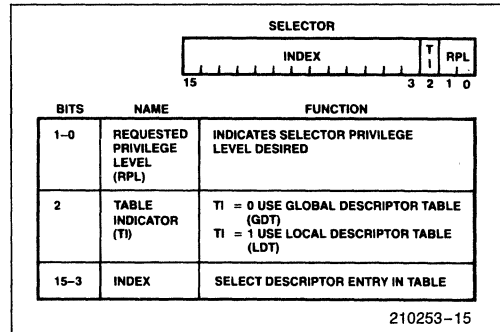


Figure 15. Selector Fields

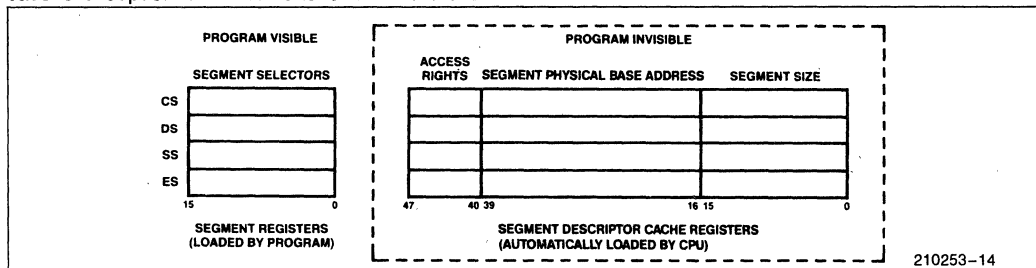


Figure 14. Descriptor Cache Registers

LOCAL AND GLOBAL DESCRIPTOR TABLES

Two tables of descriptors, called descriptor tables, contain all descriptors accessible by a task at any given time. A descriptor table is a linear array of up to 8192 descriptors. The upper 13 bits of the selector value are an index into a descriptor table. Each table has a 24-bit base register to locate the descriptor table in physical memory and a 16-bit limit register that confine descriptor access to the defined limits of the table as shown in Figure 16. A restartable exception (13) will occur if an attempt is made to reference a descriptor outside the table limits.

One table, called the Global Descriptor table (GDT), contains descriptors available to all tasks. The other table, called the Local Descriptor Table (LDT), contains descriptors that can be private to a task. Each task may have its own private LDT. The GDT may contain all descriptor types except interrupt and trap descriptors. The LDT may contain only segment, task gate, and call gate descriptors. A segment cannot be accessed by a task if its segment descriptor does not exist in either descriptor table at the time of access.

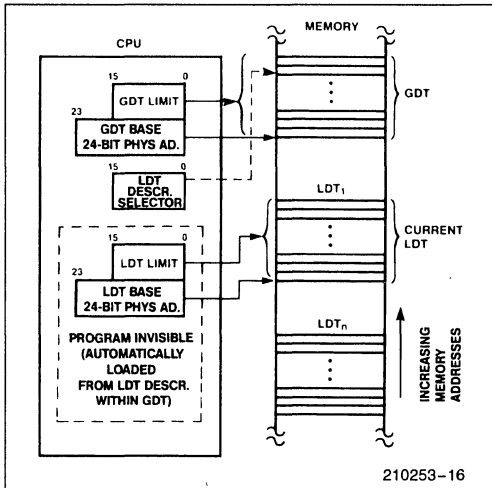


Figure 16. Local and Global Descriptor Table Definition

The LGDT and LLDT instructions load the base and limit of the global and local descriptor tables. LGDT and LLDT are privileged, i.e. they may only be executed by trusted programs operating at level 0. The LGDT instruction loads a six byte field containing the 16-bit table limit and 24-bit physical base address of the Global Descriptor Table as shown in Figure 17. The LLDT instruction loads a selector which refers to a Local Descriptor Table descriptor containing the

base address and limit for an LDT, as shown in Figure 12.

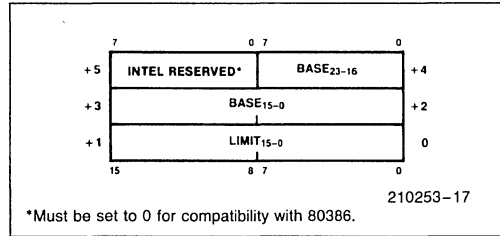


Figure 17. Global Descriptor Table and Interrupt Descriptor Table Data Type

INTERRUPT DESCRIPTOR TABLE

The protected mode 80286 has a third descriptor table, called the Interrupt Descriptor Table (IDT) (see Figure 18), used to define up to 256 interrupts. It may contain only task gates, interrupt gates and trap gates. The IDT (Interrupt Descriptor Table) has a 24-bit physical base and 16-bit limit register in the CPU. The privileged LIDT instruction loads these registers with a six byte value of identical form to that of the LGDT instruction (see Figure 17 and Protected Mode Initialization).

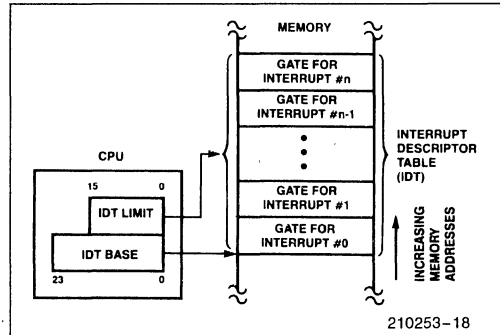
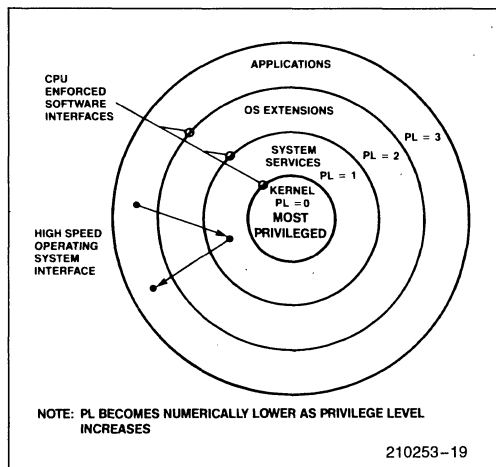


Figure 18. Interrupt Descriptor Table Definition

References to IDT entries are made via INT instructions, external interrupt vectors, or exceptions. The IDT must be at least 256 bytes in size to allocate space for all reserved interrupts.

Privilege

The 80286 has a four-level hierarchical privilege system which controls the use of privileged instructions and access to descriptors (and their associated segments) within a task. Four-level privilege, as shown in Figure 19, is an extension of the user/supervisor mode commonly found in minicomputers. The privilege levels are numbered 0 through 3. Level 0 is the



most privileged level. Privilege levels provide protection within a task. (Tasks are isolated by providing private LDT's for each task.) Operating system routines, interrupt handlers, and other system software can be included and protected within the virtual address space of each task using the four levels of privilege. Each task in the system has a separate stack for each of its privilege levels.

Tasks, descriptors, and selectors have a privilege level attribute that determines whether the descriptor may be used. Task privilege effects the use of instructions and descriptors. Descriptor and selector privilege only effect access to the descriptor.

TASK PRIVILEGE

A task always executes at one of the four privilege levels. The task privilege level at any specific instant is called the Current Privilege Level (CPL) and is defined by the lower two bits of the CS register. CPL cannot change during execution in a single code segment. A task's CPL may only be changed by control transfers through gate descriptors to a new code segment (See Control Transfer). Tasks begin executing at the CPL value specified by the code segment selector within TSS when the task is initiated via a task switch operation (See Figure 20). A task executing at Level 0 can access all data segments defined in the GDT and the task's LDT and is considered the most trusted level. A task executing a Level 3 has the most restricted access to data and is considered the least trusted level.

DESCRIPTOR PRIVILEGE

Descriptor privilege is specified by the Descriptor Privilege Level (DPL) field of the descriptor access byte. DPL specifies the least trusted task privilege level (CPL) at which a task may access the descrip-

tor. Descriptors with DPL = 0 are the most protected. Only tasks executing at privilege level 0 (CPL = 0) may access them. Descriptors with DPL = 3 are the least protected (i.e. have the least restricted access) since tasks can access them when CPL = 0, 1, 2, or 3. This rule applies to all descriptors, except LDT descriptors.

SELECTOR PRIVILEGE

Selector privilege is specified by the Requested Privilege Level (RPL) field in the least significant two bits of a selector. Selector RPL may establish a less trusted privilege level than the current privilege level for the use of a selector. This level is called the task's effective privilege level (EPL). RPL can only reduce the scope of a task's access to data with this selector. A task's effective privilege is the numeric maximum of RPL and CPL. A selector with RPL = 0 imposes no additional restriction on its use while a selector with RPL = 3 can only refer to segments at privilege Level 3 regardless of the task's CPL. RPL is generally used to verify that pointer parameters passed to a more trusted procedure are not allowed to use data at a more privileged level than the caller (refer to pointer testing instructions).

Descriptor Access and Privilege Validation

Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL. The two basic types of segment accesses are control transfer (selectors loaded into CS) and data (selectors loaded into DS, ES or SS).

DATA SEGMENT ACCESS

Instructions that load selectors into DS and ES must refer to a data segment descriptor or readable code segment descriptor. The CPL of the task and the RPL of the selector must be the same as or more privileged (numerically equal to or lower than) than the descriptor DPL. In general, a task can only access data segments at the same or less privileged levels than the CPL or RPL (whichever is numerically higher) to prevent a program from accessing data it cannot be trusted to use.

An exception to the rule is a readable conforming code segment. This type of code segment can be read from any privilege level.

If the privilege checks fail (e.g. DPL is numerically less than the maximum of CPL and RPL) or an incorrect type of descriptor is referenced (e.g. gate de-

scriptor or execute only code segment) exception 13 occurs. If the segment is not present, exception 11 is generated.

Instructions that load selectors into SS must refer to data segment descriptors for writable data segments. The descriptor privilege (DPL) and RPL must equal CPL. All other descriptor types or a privilege level violation will cause exception 13. A not present fault causes exception 12.

CONTROL TRANSFER

Four types of control transfer can occur when a selector is loaded into CS by a control transfer operation (see Table 10). Each transfer type can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules (e.g. JMP through a call gate or RET to a Task State Segment) will cause exception 13.

The ability to reference a descriptor for control transfer is also subject to rules of privilege. A CALL or JUMP instruction may only reference a code segment descriptor with DPL equal to the task CPL or a conforming segment with DPL of equal or greater privilege than CPL. The RPL of the selector used to reference the code descriptor must have as much privilege as CPL.

RET and IRET instructions may only reference code segment descriptors with descriptor privilege equal to or less privileged than the task CPL. The selector loaded into CS is the return address from the stack. After the return, the selector RPL is the task's new CPL. If CPL changes, the old stack pointer is popped after the return address.

When a JMP or CALL references a Task State Segment descriptor, the descriptor DPL must be the same or less privileged than the task's CPL. Refer-

ence to a valid Task State Segment descriptor causes a task switch (see Task Switch Operation). Reference to a Task State Segment descriptor at a more privileged level than the task's CPL generates exception 13.

When an instruction or interrupt references a gate descriptor, the gate DPL must have the same or less privilege than the task CPL. If DPL is at a more privileged level than CPL, exception 13 occurs. If the destination selector contained in the gate references a code segment descriptor, the code segment descriptor DPL must be the same or more privileged than the task CPL. If not, Exception 13 is issued. After the control transfer, the code segment descriptors DPL is the task's new CPL. If the destination selector in the gate references a task state segment, a task switch is automatically performed (see Task Switch Operation).

The privilege rules on control transfer require:

- JMP or CALL direct to a code segment (code segment descriptor) can only be to a conforming segment with DPL of equal or greater privilege than CPL or a non-conforming segment at the same privilege level.
- interrupts within the task or calls that may change privilege levels, can only transfer control through a gate at the same or a less privileged level than CPL to a code segment at the same or more privileged level than CPL.
- return instructions that don't switch tasks can only return control to a code segment at the same or less privileged level.
- task switch can be performed by a call, jump or interrupt which references either a task gate or task state segment at the same or less privileged level.

Table 10. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL.	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
Task Switch	CALL, JMP	Task State Segment	GDT
	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag word) = 0

**NT (Nested Task bit of flag word) = 1

PRIVILEGE LEVEL CHANGES

Any control transfer that changes CPL within the task, causes a change of stacks as part of the operation. Initial values of SS:SP for privilege levels 0, 1, and 2 are kept in the task state segment (refer to Task Switch Operation). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and SP registers and the previous stack pointer is pushed onto the new stack.

When returning to the original privilege level, its stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words, as specified in the gate, are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

Protection

The 80286 includes mechanisms to protect critical instructions that affect the CPU execution state (e.g. HLT) and code or data segments from improper usage. These protection mechanisms are grouped into three forms:

Restricted *usage* of segments (e.g. no write allowed to read-only data segments). The only segments available for use are defined by descriptors in the Local Descriptor Table (LDT) and Global Descriptor Table (GDT).

Restricted *access* to segments via the rules of privilege and descriptor usage.

Privileged instructions or operations that may only be executed at certain privilege levels as determined by the CPL and I/O Privilege Level (IOPL). The IOPL is defined by bits 14 and 13 of the flag word.

These checks are performed for all instructions and can be split into three categories: segment load checks (Table 11), operand reference checks (Table 12), and privileged instruction checks (Table 13). Any violation of the rules shown will result in an exception. A not-present exception related to the stack segment causes exception 12.

The IRET and POPF instructions do not perform some of their defined functions if CPL is not of sufficient privilege (numerically small enough). Precisely these are:

- The IF bit is not changed if $CPL > IOPL$.
- The IOPL field of the flag word is not changed if $CPL > 0$.

No exceptions or other indication are given when these conditions occur.

Table 11
Segment Register Load Checks

Error Description	Exception Number
Descriptor table limit exceeded	13
Segment descriptor not-present	11 or 12
Privilege rules violated	13
Invalid descriptor/segment type segment register load: —Read only data segment load to SS —Special Control descriptor load to DS, ES, SS —Execute only segment load to DS, ES, SS —Data segment load to CS —Read/Execute code segment load to SS	13

Table 12. Operand Reference Checks

Error Description	Exception Number
Write into code segment	13
Read from execute-only code segment	13
Write to read-only data segment	13
Segment limit exceeded ¹	12 or 13

NOTE:

Carry out in offset calculations is ignored.

Table 13. Privileged Instruction Checks

Error Description	Exception Number
$CPL \neq 0$ when executing the following instructions: LIDT, LLDT, LGDT, LTR, LMSW, CTS, HLT	13
$CPL > IOPL$ when executing the following instructions: INS, IN, OUTS, OUT, STI, CLI, LOCK	13

EXCEPTIONS

The 80286 detects several types of exceptions and interrupts, in protected mode (see Table 14). Most are restartable after the exceptional condition is removed. Interrupt handlers for most exceptions can read an error code, pushed on the stack after the return address, that identifies the selector involved (0 if none). The return address normally points to the failing instruction, including all leading prefixes. For a processor extension segment overrun exception, the return address will not point at the ESC instruction that caused the exception; however, the processor extension registers may contain the address of the failing instruction.

Table 14. Protected Mode Exceptions

Interrupt Vector	Function	Return Address At Falling Instruction?	Always Restartable?	Error Code on Stack?
8	Double exception detected	Yes	No ²	Yes
9	Processor extension segment overrun	No	No ²	No
10	Invalid task state segment	Yes	Yes	Yes
11	Segment not present	Yes	Yes	Yes
12	Stack segment overrun or stack segment not present	Yes	Yes ¹	Yes
13	General protection	Yes	No ²	Yes

NOTE:

- When a PUSHA or POPA instruction attempts to wrap around the stack segment, the machine state after the exception will not be restartable because stack segment wrap around is not permitted. This condition is identified by the value of the saved SP being either 0000(H), 0001(H), FFFE(H), or FFFF(H).
- These exceptions indicate a violation to privilege rules or usage rules has occurred. Restart is generally not attempted under those conditions.

These exceptions indicate a violation to privilege rules or usage rules has occurred. Restart is generally not attempted under those conditions.

All these checks are performed for all instructions and can be split into three categories: segment load checks (Table 11), operand reference checks (Table 12), and privileged instruction checks (Table 13). Any violation of the rules shown will result in an exception. A not-present exception causes exception 11 or 12 and is restartable.

Special Operations

TASK SWITCH OPERATION

The 80286 provides a built-in task switch operation which saves the entire 80286 execution state (registers, address space, and a link to the previous task), loads a new execution state, and commences execution in the new task. Like gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS) or task gate descriptor in the GDT or LDT. An INT n instruction, exception, or external interrupt may also invoke the task switch operation by selecting a task gate descriptor in the associated IDT descriptor entry.

The TSS descriptor points at a segment (see Figure 20) containing the entire 80286 execution state while a task gate descriptor contains a TSS selector. The limit field of the descriptor must be >002B(H).

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 80286 called the Task Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector.

The IRET instruction is used to return control to the task that called the current task or was interrupted. Bit 14 in the flag register is called the Nested Task (NT) bit. It controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular current task by popping values off the stack; when NT = 1, IRET performs a task switch operation back to the previous task.

When a CALL, JMP, or INT instruction initiates a task switch, the old (except for case of JMP) and new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. NT may also be set or cleared by POPF or IRET instructions.

The task state segment is marked busy by changing the descriptor type field from Type 1 to Type 3. Use of a selector that references a busy task state segment causes Exception 13.

PROCESSOR EXTENSION CONTEXT SWITCHING

The context of a processor extension (such as the 80287 numerics processor) is not changed by the task switch operation. A processor extension context need only be changed when a different task attempts to use the processor extension (which still contains the context of a previous task). The 80286 detects the first use of a processor extension after a task switch by causing the processor extension not present exception (7). The interrupt handler may then decide whether a context change is necessary.

Whenever the 80286 switches tasks, it sets the Task Switched (TS) bit of the MSW. TS indicates that a processor extension context may belong to a different task than the current one. The processor extension not present exception (7) will occur when attempting to execute an ESC or WAIT instruction if TS = 1 and a processor extension is present (MP = 1 in MSW).

POINTER TESTING INSTRUCTIONS

The 80286 provides several instructions to speed pointer testing and consistency checks for maintaining system integrity (see Table 15). These instruc-

tions use the memory management hardware to verify that a selector value refers to an appropriate segment without risking an exception. A condition flag (ZF) indicates whether use of the selector or segment will cause an exception.

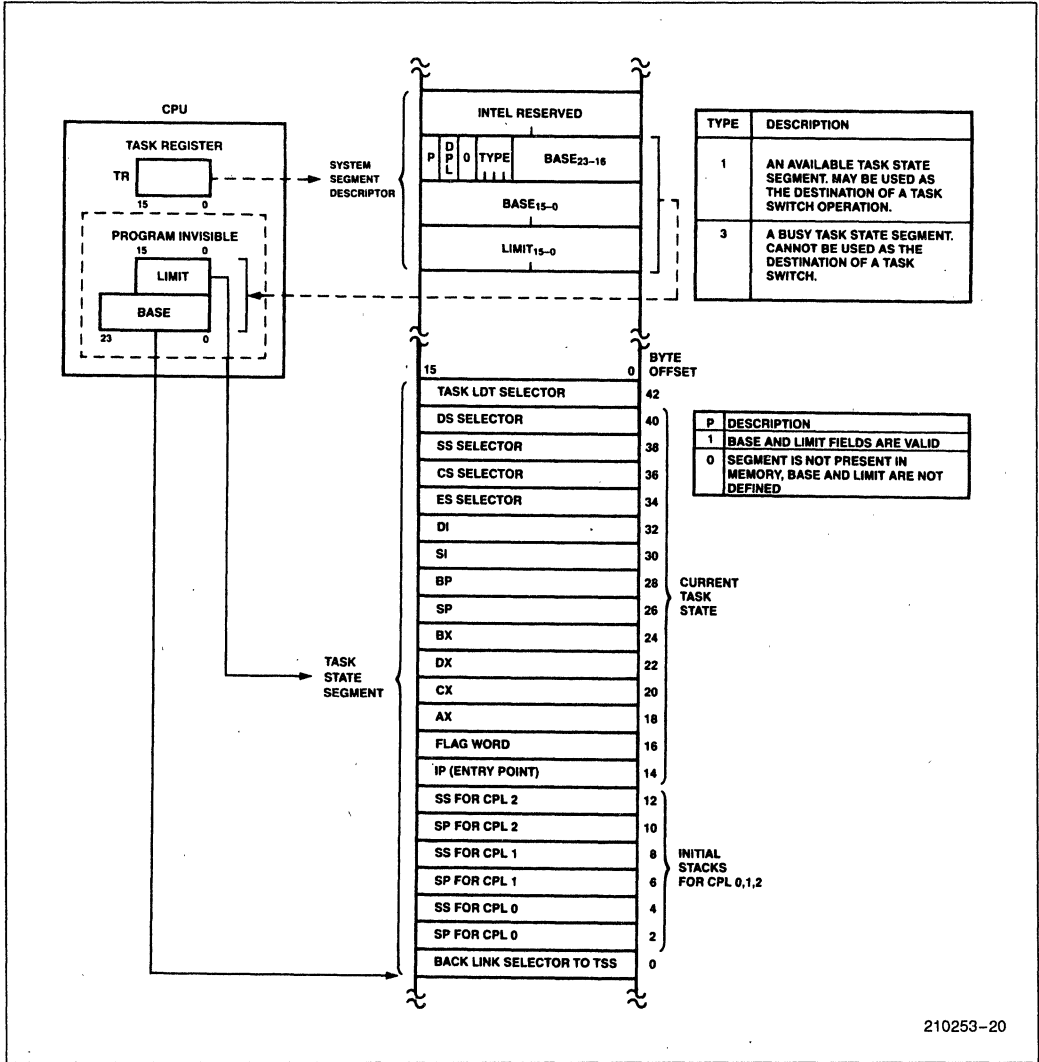


Figure 20. Task State Segment and TSS Registers

Table 15. 80286 Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed by ARPL.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

DOUBLE FAULT AND SHUTDOWN

If two separate exceptions are detected during a single instruction execution, the 80286 performs the double fault exception (8). If an execution occurs during processing of the double fault exception, the 80286 will enter shutdown. During shutdown no further instructions or exceptions are processed. Either NMI (CPU remains in protected mode) or RESET (CPU exits protected mode) can force the 80286 out of shutdown. Shutdown is externally signalled via a HALT bus operation with A_1 LOW.

PROTECTED MODE INITIALIZATION

The 80286 initially executes in real address mode after RESET. To allow initialization code to be placed at the top of physical memory, $A_{23}-A_{20}$ will be HIGH when the 80286 performs memory references relative to the CS register until CS is changed. $A_{23}-A_{20}$ will be zero for references to the DS, ES, or SS segments. Changing CS in real address mode will force $A_{23}-A_{20}$ LOW whenever CS is used again. The initial CS:IP value of F000:FFF0 provides 64K bytes of code space for initialization code without changing CS.

Protected mode operation requires several registers to be initialized. The GDT and IDT base registers must refer to a valid GDT and IDT. After executing the LMSW instruction to set PE, the 80286 must im-

mediately execute an intra-segment JMP instruction to clear the instruction queue of instructions decoded in real address mode.

To force the 80286 CPU registers to match the initial protected mode state assumed by software, execute a JMP instruction with a selector referring to the initial TSS used in the system. This will load the task register, local descriptor table register, segment registers and initial general register state. The TR should point at a valid TSS since any task switch operation involves saving the current task state.

SYSTEM INTERFACE

The 80286 system interface appears in two forms: a local bus and a system bus. The local bus consists of address, data, status, and control signals at the pins of the CPU. A system bus is any buffered version of the local bus. A system bus may also differ from the local bus in terms of coding of status and control lines and/or timing and loading of signals. The 80286 family includes several devices to generate standard system buses such as the IEEE 796 standard MULTIBUS.

Bus Interface Signals and Timing

The 80286 microsystem local bus interfaces the 80286 to local memory and I/O components. The interface has 24 address lines, 16 data lines, and 8 status and control signals.

The 80286 CPU, 82C284 clock generator, 82C288 bus controller, trapeceivers, and latches provide a buffered and decoded system bus interface. The 82C284 generates the system clock and synchronizes READY and RESET. The 82C288 converts bus operation status encoded by the 80286 into command and bus control signals. These components can provide the timing and electrical power drive levels required for most system bus interfaces including the Multibus.

Physical Memory and I/O Interface

A maximum of 16 megabytes of physical memory can be addressed in protected mode. One megabyte can be addressed in real address mode. Memory is accessible as bytes or words. Words consist of any two consecutive bytes addressed with the least significant byte stored in the lowest address.

Byte transfers occur on either half of the 16-bit local data bus. Even bytes are accessed over D_{7-0} while odd bytes are transferred over D_{15-8} . Even-addressed words are transferred over D_{15-0} in one bus cycle, while odd-addressed word require *two* bus operations. The first transfers data on D_{15-8} , and the second transfers data on D_{7-0} . Both byte data transfers occur automatically, transparent to software.

Two bus signals, A_0 and \overline{BHE} , control transfers over the lower and upper halves of the data bus. Even address byte transfers are indicated by A_0 LOW and \overline{BHE} HIGH. Odd address byte transfers are indicated by A_0 HIGH and \overline{BHE} LOW. Both A_0 and \overline{BHE} are LOW for even address word transfers.

The I/O address space contains 64K addresses in both modes. The I/O space is accessible as either bytes or words, as is memory. Byte wide peripheral devices may be attached to either the upper or lower byte of the data bus. Byte-wide I/O devices attached to the upper data byte (D_{15-8}) are accessed with odd I/O addresses. Devices on the lower data byte are accessed with even I/O addresses. An interrupt controller such as Intel's 8259A must be connected to the lower data byte (D_{7-0}) for proper return of the interrupt vector.

Bus Operation

The 80286 uses a double frequency system clock (CLK input) to control bus timing. All signals on the local bus are measured relative to the system CLK input. The CPU divides the system clock by 2 to produce the internal processor clock, which determines bus state. Each processor clock is composed of two system clock cycles named phase 1 and phase 2. The 82C284 clock generator output (PCLK) identifies the next phase of the processor clock. (See Figure 21.)

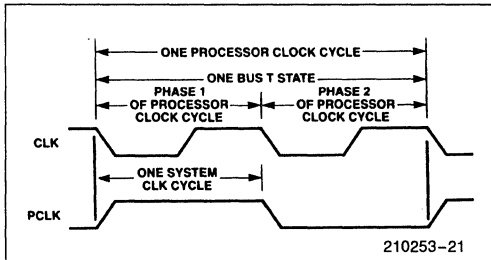


Figure 21. System and Processor Clock Relationships

Six types of bus operations are supported; memory read, memory write, I/O read, I/O write, interrupt acknowledge, and halt/shutdown. Data can be transferred at a maximum rate of one word per two processor clock cycles.

The 80286 bus has three basic states: idle (T_i), send status (T_s), and perform command (T_c). The 80286 CPU also has a fourth local bus state called hold (T_h). T_h indicates that the 80286 has surrendered control of the local bus to another bus master in response to a HOLD request.

Each bus state is one processor clock long. Figure 22 shows the four 80286 local bus states and allowed transitions.

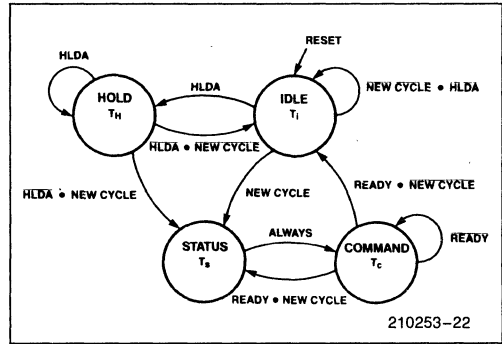


Figure 22. 80286 Bus States

Bus States

The idle (T_i) state indicates that no data transfers are in progress or requested. The first active state T_s is signaled by status line S_1 or S_0 going LOW and identifying phase 1 of the processor clock. During T_s , the command encoding, the address, and data (for a write operation) are available on the 80286 output pins. The 82C288 bus controller decodes the status signals and generates Multibus compatible read/write command and local transceiver control signals.

After T_s , the perform command (T_c) state is entered. Memory or I/O devices respond to the bus operation during T_c , either transferring read data to the CPU or accepting write data. T_c states may be repeated as often as necessary to assure sufficient time for the memory or I/O device to respond. The \overline{READY} signal determines whether T_c is repeated. A repeated T_c state is called a wait state.

During hold (T_h), the 80286 will float all address, data, and status output pins enabling another bus master to use the local bus. The 80286 HOLD input signal is used to place the 80286 into the T_h state. The 80286 HLDA output signal indicates that the CPU has entered T_h .

Pipelined Addressing

The 80286 uses a local bus interface with pipelined timing to allow as much time as possible for data access. Pipelined timing allows a new bus operation to be initiated every two processor cycles, while allowing each individual bus operation to last for three processor cycles.

The timing of the address outputs is pipelined such that the address of the next bus operation becomes available during the current bus operation. Or in other words, the first clock of the next bus operation is overlapped with the last clock of the current bus operation. Therefore, address decode and routing logic can operate in advance of the next bus operation.

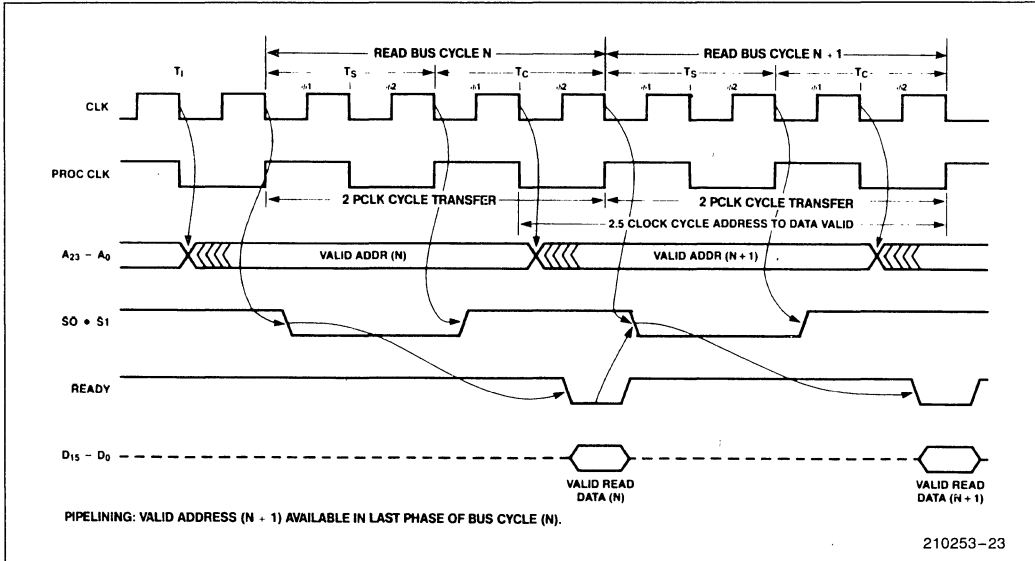


Figure 23. Basic Bus Cycle

External address latches may hold the address stable for the entire bus operation, and provide additional AC and DC buffering.

The 80286 does not maintain the address of the current bus operation during all T_c states. Instead, the address for the next bus operation may be emitted during phase 2 of any T_c . The address remains valid during phase 1 of the first T_c to guarantee hold time, relative to ALE, for the address latch inputs.

Bus Control Signals

The 82C288 bus controller provides control signals; address latch enable (ALE), Read/Write commands, data transmit/receive (DT/R), and data enable (DEN) that control the address latches, data transceivers, write enable, and output enable for memory and I/O systems.

The Address Latch Enable (ALE) output determines when the address may be latched. ALE provides at least one system CLK period of address hold time from the end of the previous bus operation until the address for the next bus operation appears at the latch outputs. This address hold time is required to support MULTIBUS and common memory systems.

The data bus transceivers are controlled by 82C288 outputs Data Enable (DEN) and Data Transmit/Receive (DT/R). DEN enables the data transceivers; while DT/R controls trceiver direction. DEN and DT/R are timed to prevent bus contention between the bus master, data bus transceivers, and system data bus transceivers.

Command Timing Controls

Two system timing customization options, command extension and command delay, are provided on the 80286 local bus.

Command extension allows additional time for external devices to respond to a command and is analogous to inserting wait states on the 8086. External logic can control the duration of any bus operation such that the operation is only as long as necessary. The READY input signal can extend any bus operation for as long as necessary.

Command delay allows an increase of address or write data setup time to system bus command active for any bus operation by delaying when the system bus command becomes active. Command delay is controlled by the 82C288 CMDLY input. After T_s , the bus controller samples CMDLY at each falling edge of CLK. If CMDLY is HIGH, the 82C288 will not activate the command signal. When CMDLY is LOW, the 82C288 will activate the command signal. After the command becomes active, the CMDLY input is not sampled.

When a command is delayed, the available response time from command active to return read data or accept write data is less. To customize system bus timing, an address decoder can determine which bus operations require delaying the command. The CMDLY input does not affect the timing of ALE, DEN, or DT/R.

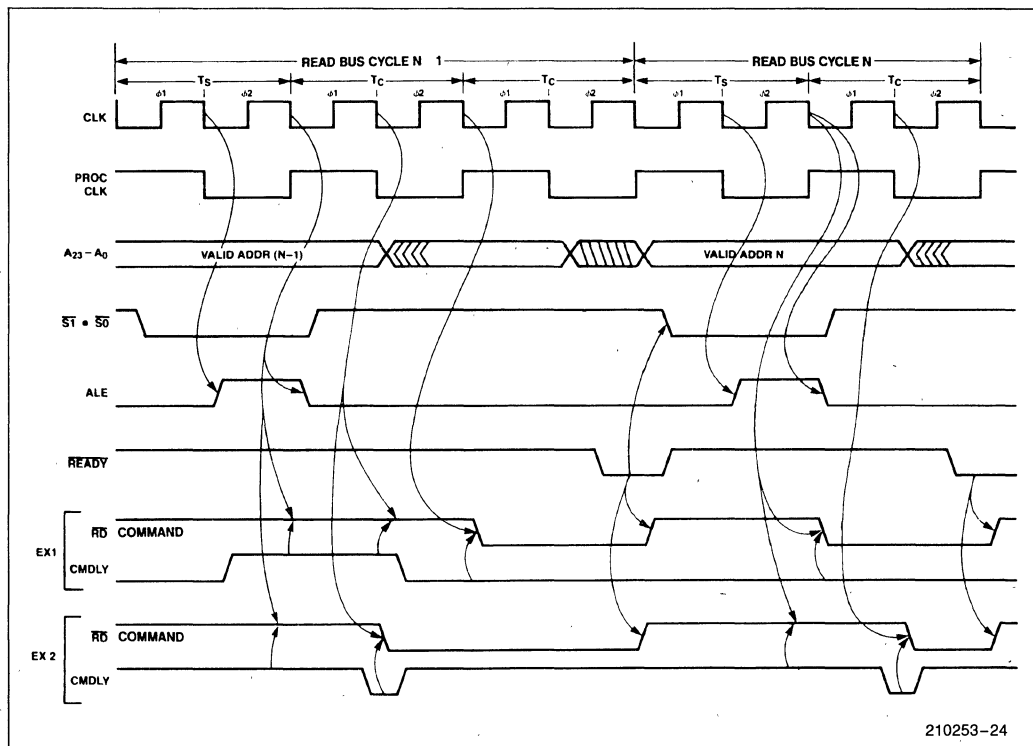


Figure 24. CMDLY Controls the Leading Edge of Command Signal

Figure 24 illustrates four uses of CMDLY. Example 1 shows delaying the read command two system CLKs for cycle N-1 and no delay for cycle N, and example 2 shows delaying the read command one system CLK for cycle N-1 and one system CLK delay for cycle N.

Bus Cycle Termination

At maximum transfer rates, the 80286 bus alternates between the status and command states. The bus status signals become inactive after T_S so that they may correctly signal the start of the next bus operation after the completion of the current cycle. No external indication of T_C exists on the 80286 local bus. The bus master and bus controller enter T_C directly after T_S and continue executing T_C cycles until terminated by $\overline{\text{READY}}$.

$\overline{\text{READY}}$ Operation

The current bus master and 82C288 bus controller terminate each bus operation simultaneously to achieve maximum bus operation bandwidth. Both are informed in advance by $\overline{\text{READY}}$ active (open-collector output from 82C284) which identifies the last T_C cycle of the current bus operation. The bus master and bus controller must see the same sense

of the $\overline{\text{READY}}$ signal, thereby requiring $\overline{\text{READY}}$ be synchronous to the system clock.

Synchronous Ready

The 82C284 clock generator provides $\overline{\text{READY}}$ synchronization from both synchronous and asynchronous sources (see Figure 25). The synchronous ready input ($\overline{\text{SRDY}}$) of the clock generator is sampled with the falling edge of CLK at the end of phase 1 of each T_C . The state of $\overline{\text{SRDY}}$ is then broadcast to the bus master and bus controller via the $\overline{\text{READY}}$ output line.

Asynchronous Ready

Many systems have devices or subsystems that are asynchronous to the system clock. As a result, their ready outputs cannot be guaranteed to meet the 82C284 $\overline{\text{SRDY}}$ setup and hold time requirements. But the 82C284 asynchronous ready input ($\overline{\text{ARDY}}$) is designed to accept such signals. The $\overline{\text{ARDY}}$ input is sampled at the beginning of each T_C cycle by 82C284 synchronization logic. This provides one system CLK cycle time to resolve its value before broadcasting it to the bus master and bus controller.

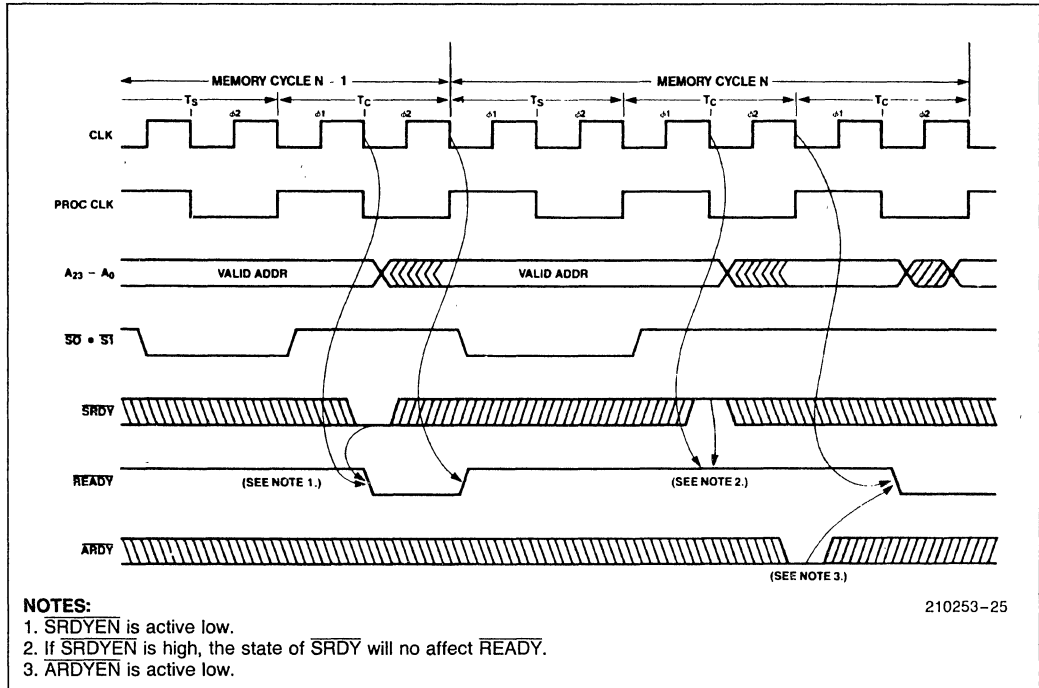


Figure 25. Synchronous and Asynchronous Ready

\overline{ARDY} or \overline{ARDYEN} must be HIGH at the end of T_S . \overline{ARDY} cannot be used to terminate bus cycle with no wait states.

Each ready input of the 82C284 has an enable pin (\overline{SRDYEN} and \overline{ARDYEN}) to select whether the current bus operation will be terminated by the synchronous or asynchronous ready. Either of the ready inputs may terminate a bus operation. These enable inputs are active low and have the same timing as their respective ready inputs. Address decode logic usually selects whether the current bus operation should be terminated by \overline{ARDY} or \overline{SRDY} .

Data Bus Control

Figures 26, 27, and 28 show how the DT/\overline{R} , DEN , data bus, and address signals operate for different combinations of read, write, and idle bus operations. DT/\overline{R} goes active (LOW) for a read operation. DT/\overline{R} remains HIGH before, during, and between write operations.

The data bus is driven with write data during the second phase of T_S . The delay in write data timing allows the read data drivers, from a previous read cycle, sufficient time to enter 3-state OFF before the 80286 CPU begins driving the local data bus for write operations. Write data will always remain valid for one system clock past the last T_C to provide sufficient hold time for Multibus or other similar memory or I/O systems. During write-read or write-idle sequences the data bus enters 3-state OFF during the second phase of the processor cycle after the last T_C . In a write-write sequence the data bus does not enter 3-state OFF between T_C and T_S .

Bus Usage

The 80286 local bus may be used for several functions: instruction data transfers, data transfers by other bus masters, instruction fetching, processor extension data transfers, interrupt acknowledge, and halt/shutdown. This section describes local bus activities which have special signals or requirements.

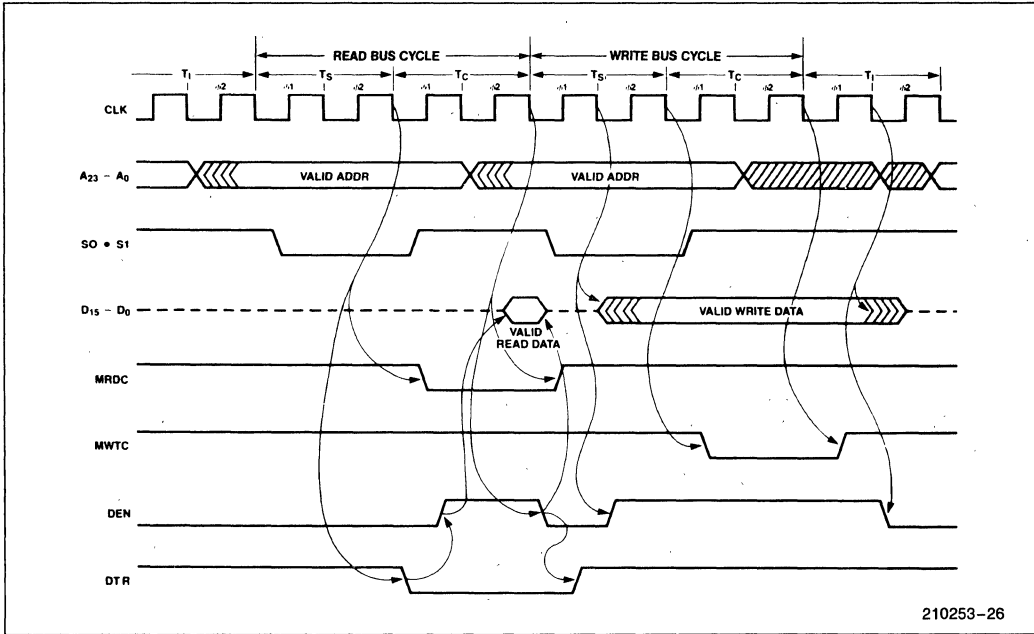


Figure 26. Back to Back Read-Write Cycles

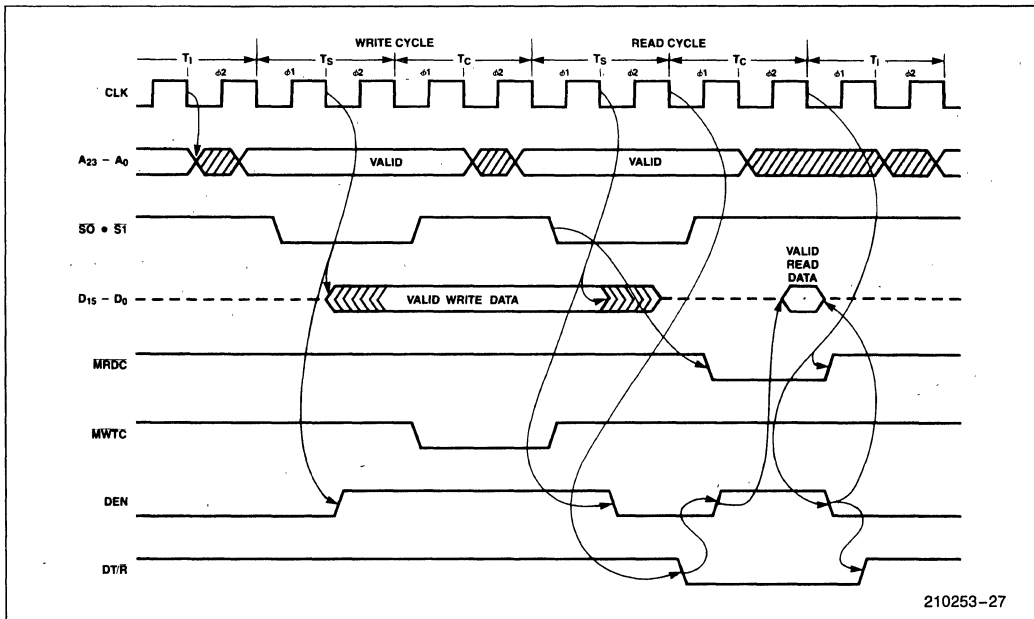


Figure 27. Back to Back Write-Read Cycles

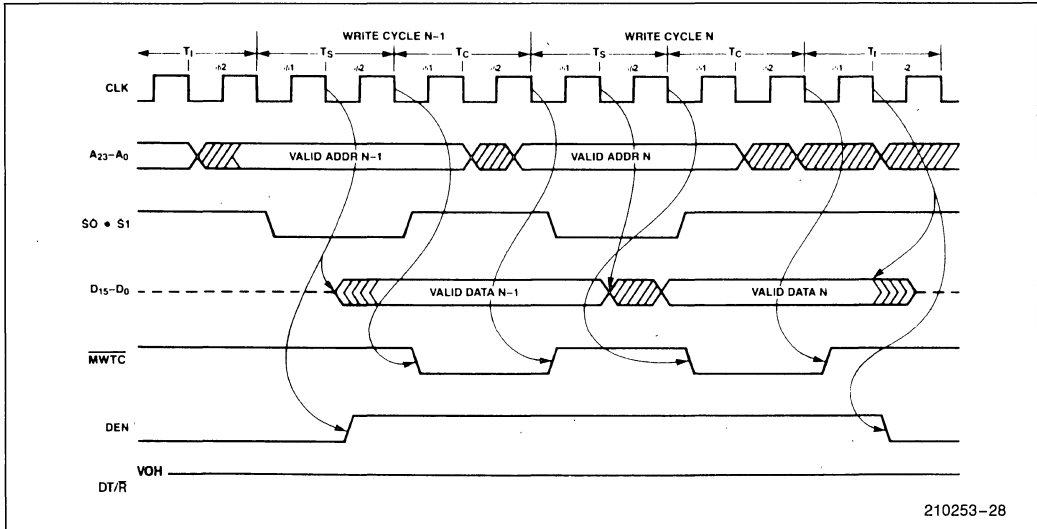


Figure 28. Back to Back Write-Write Cycles

HOLD and HLDA

HOLD AND HLDA allow another bus master to gain control of the local bus by placing the 80286 bus into the T_h state. The sequence of events required to pass control between the 80286 and another local bus master are shown in Figure 29.

In this example, the 80286 is initially in the T_h state as signaled by HLDA being active. Upon leaving T_h , as signaled by HLDA going inactive, a write operation is started. During the write operation another local bus master requests the local bus from the 80286 as shown by the HOLD signal. After completing the write operation, the 80286 performs one T_i bus cycle, to guarantee write data hold time, then enters T_h as signaled by HLDA going active.

The \overline{CMDLY} signal and \overline{ARDY} ready are used to start and stop the write bus command, respectively. Note that \overline{SRDY} must be inactive or disabled by \overline{SRDYEN} to guarantee \overline{ARDY} will terminate the cycle.

HOLD must not be active during the time from the leading edge of RESET until 34 CLKs following the trailing edge of RESET.

Lock

The CPU asserts an active lock signal during Interrupt-Acknowledge cycles, the XCHG instruction, and during some descriptor accesses. Lock is also asserted when the LOCK prefix is used. The LOCK prefix may be used with the following ASM-286 assembly instructions; MOVS, INS, and OUTS. For bus

cycles other than Interrupt-Acknowledge cycles, Lock will be active for the first and subsequent cycles of a series of cycles to be locked. Lock will not be shown active during the last cycle to be locked. For the next-to-last cycle, Lock will become inactive at the end of the first T_c regardless of the number of wait-states inserted. For Interrupt-Acknowledge cycles, Lock will be active for each cycle, and will become inactive at the end of the first T_c for each cycle regardless of the number of wait-states inserted.

Instruction Fetching

The 80286 Bus Unit (BU) will fetch instructions ahead of the current instruction being executed. This activity is called prefetching. It occurs when the local bus would otherwise be idle and obeys the following rules:

A prefetch bus operation starts when at least two bytes of the 6-byte prefetch queue are empty.

The prefetcher normally performs word prefetches independent of the byte alignment of the code segment base in physical memory.

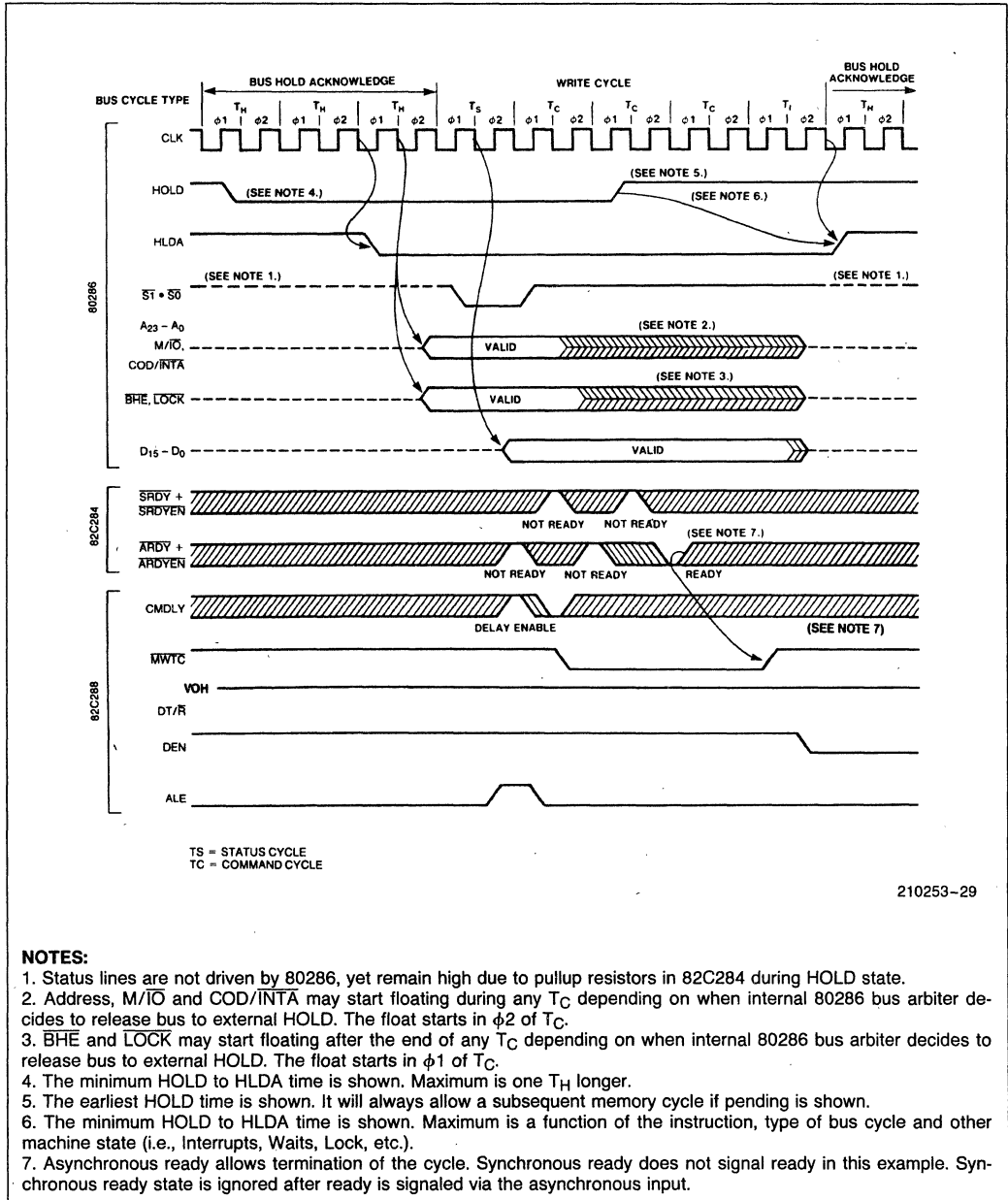
The prefetcher will perform only a byte code fetch operation for control transfers to an instruction beginning on a numerically odd physical address.

Prefetching stops whenever a control transfer or HLT instruction is decoded by the IU and placed into the instruction queue.

In real address mode, the prefetcher may fetch up to 6 bytes beyond the last control transfer or HLT instruction in a code segment.

In protected mode, the prefetcher will never cause a segment overrun exception. The prefetcher stops at the last physical memory word of the code segment. Exception 13 will occur if the program attempts to execute beyond the last full instruction in the code segment.

If the last byte of a code segment appears on an even physical memory address, the prefetcher will read the next physical byte of memory (perform a word code fetch). The value of this byte is ignored and any attempt to execute it causes exception 13.



NOTES:

1. Status lines are not driven by 80286, yet remain high due to pullup resistors in 82C284 during HOLD state.
2. Address, M/I \bar{O} and COD/INT \bar{A} may start floating during any T_C depending on when internal 80286 bus arbiter decides to release bus to external HOLD. The float starts in $\phi 2$ of T_C.
3. \bar{BHE} and \bar{LOCK} may start floating after the end of any T_C depending on when internal 80286 bus arbiter decides to release bus to external HOLD. The float starts in $\phi 1$ of T_C.
4. The minimum HOLD to HLDA time is shown. Maximum is one T_H longer.
5. The earliest HOLD time is shown. It will always allow a subsequent memory cycle if pending is shown.
6. The minimum HOLD to HLDA time is shown. Maximum is a function of the instruction, type of bus cycle and other machine state (i.e., interrupts, Waits, Lock, etc.).
7. Asynchronous ready allows termination of the cycle. Synchronous ready does not signal ready in this example. Synchronous ready state is ignored after ready is signaled via the asynchronous input.

Figure 29. MULTIBUS® Write Terminated by Asynchronous Ready with Bus Hold

Processor Extension Transfers

The processor extension interface uses I/O port addresses 00F8(H), 00FA(H), and 00FC(H) which are part of the I/O port address range reserved by Intel. An ESC instruction with Machine Status Word bits EM = 0 and TS = 0 will perform I/O bus operations to one or more of these I/O port addresses independent of the value of IOPL and CPL.

ESC instructions with memory references enable the CPU to accept PEREQ inputs for processor extension operand transfers. The CPU will determine the operand starting address and read/write status of the instruction. For each operand transfer, two or three bus operations are performed, one word transfer with I/O port address 00FA(H) and one or two bus operations with memory. Three bus operations are required for each word operand aligned on an odd byte address.

NOTE:

Odd-aligned numerics operands should be avoided when using an 80286 system running six or more memory-write wait states. The 80286 can generate an incorrect numerics address if all the following conditions are met:

- Two floating point (FP) instructions are fetched and in the 80286 queue.
- The first FP instruction is any floating point store except FSTSW AX.
- The second FP instruction accesses memory.
- The operand of the first instruction is aligned on an odd memory address.
- Six or more wait states are inserted during either of the last two memory write (odd aligned operands are transferred as two bytes) transfers of the first instruction.

The second FP operand's address will be incremented by one if these conditions are met. These conditions are most likely to occur in a multi-master system. For a hardware solution, contact your local Intel representative.

Commands to the numerics coprocessor should not be delayed by nine or more T-states. Excessive (nine or more) command-delays can cause the 80286 and 80287 to lose synchronization.

Interrupt Acknowledge Sequence

Figure 30 illustrates an interrupt acknowledge sequence performed by the 80286 in response to an

INTR input. An interrupt acknowledge sequence consists of two INTA bus operations. The first allows a master 8259A Programmable Interrupt Controller (PIC) to determine which if any of its slaves should return the interrupt vector. An eight bit vector is read on D0–D7 of the 80286 during the second INTA bus operation to select an interrupt handler routine from the interrupt table.

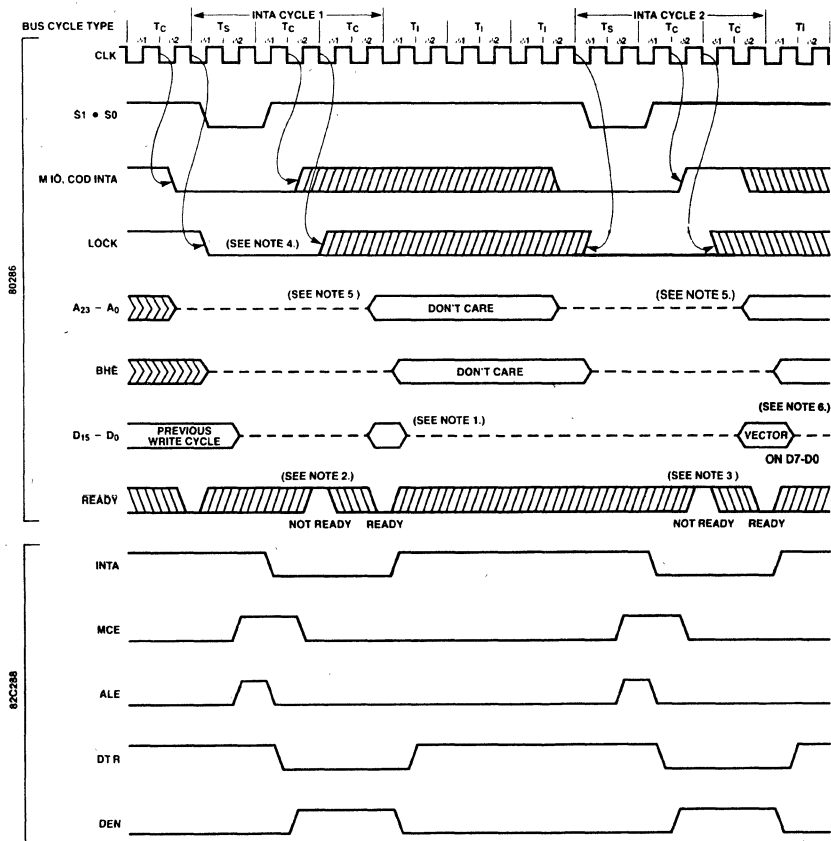
The Master Cascade Enable (MCE) signal of the 82C288 is used to enable the cascade address drivers, during INTA bus operations (See Figure 30), onto the local address bus for distribution to slave interrupt controllers via the system address bus. The 80286 emits the $\overline{\text{LOCK}}$ signal (active LOW) during T_s of the first INTA bus operation. A local bus "hold" request will not be honored until the end of the second INTA bus operation.

Three idle processor clocks are provided by the 80286 between INTA bus operations to allow for the minimum INTA to INTA time and CAS (cascade address) out delay of the 8259A. The second INTA bus operation must always have at least one extra T_c state added via logic controlling $\overline{\text{READY}}$. This is needed to meet the 8259A minimum INTA pulse width.

Local Bus Usage Priorities

The 80286 local bus is shared among several internal units and external HOLD requests. In case of simultaneous requests, their relative priorities are:

- (Highest) Any transfers which assert $\overline{\text{LOCK}}$ either explicitly (via the $\overline{\text{LOCK}}$ instruction prefix) or implicitly (i.e. some segment descriptor accesses, interrupt acknowledge sequence, or an XCHG with memory).
 - The second of the two byte bus operations required for an odd aligned word operand.
 - The second or third cycle of a processor extension data transfer.
 - Local bus request via HOLD input.
 - Processor extension data operand transfer via PEREQ input.
 - Data transfer performed by EU as part of an instruction.
- (Lowest) An instruction prefetch request from BU. The EU will inhibit prefetching two processor clocks in advance of any data transfers to minimize waiting by EU for a prefetch to finish.



210253-31

NOTES:

1. Data is ignored, upper data bus, D₈-D₁₅, should not change state during this time.
2. First INTA cycle should have at least one wait state inserted to meet 8259A minimum INTA pulse width.
3. Second INTA cycle should have at least one wait state inserted to meet 8259A minimum INTA pulse width.
4. LOCK is active for the first INTA cycle to prevent the bus arbiter from releasing the bus between INTA cycles in a multi-master system. LOCK is also active for the second INTA cycle.
5. A₂₃-A₀ exits 3-state OFF during φ₂ of the second T_C in the INTA cycle.
6. Upper data bus should not change state during this time.

Figure 30. Interrupt Acknowledge Sequence

Halt or Shutdown Cycles

The 80286 externally indicates halt or shutdown conditions as a bus operation. These conditions occur due to a HLT instruction or multiple protection exceptions while attempting to execute one instruction. A halt or shutdown bus operation is signalled when S₁, S₀ and COD/INTA are LOW and M/IO is HIGH. A₁ HIGH indicates halt, and A₁ LOW indicates shutdown. The 82C288 bus controller does

not issue ALE, nor is READY required to terminate a halt or shutdown bus operation.

During halt or shutdown, the 80286 may service PEREQ or HOLD requests. A processor extension segment overrun exception during shutdown will inhibit further service of PEREQ. Either NMI or RESET will force the 80286 out of either halt or shutdown. An INTR, if interrupts are enabled, or a processor extension segment overrun exception will also force the 80286 out of halt.

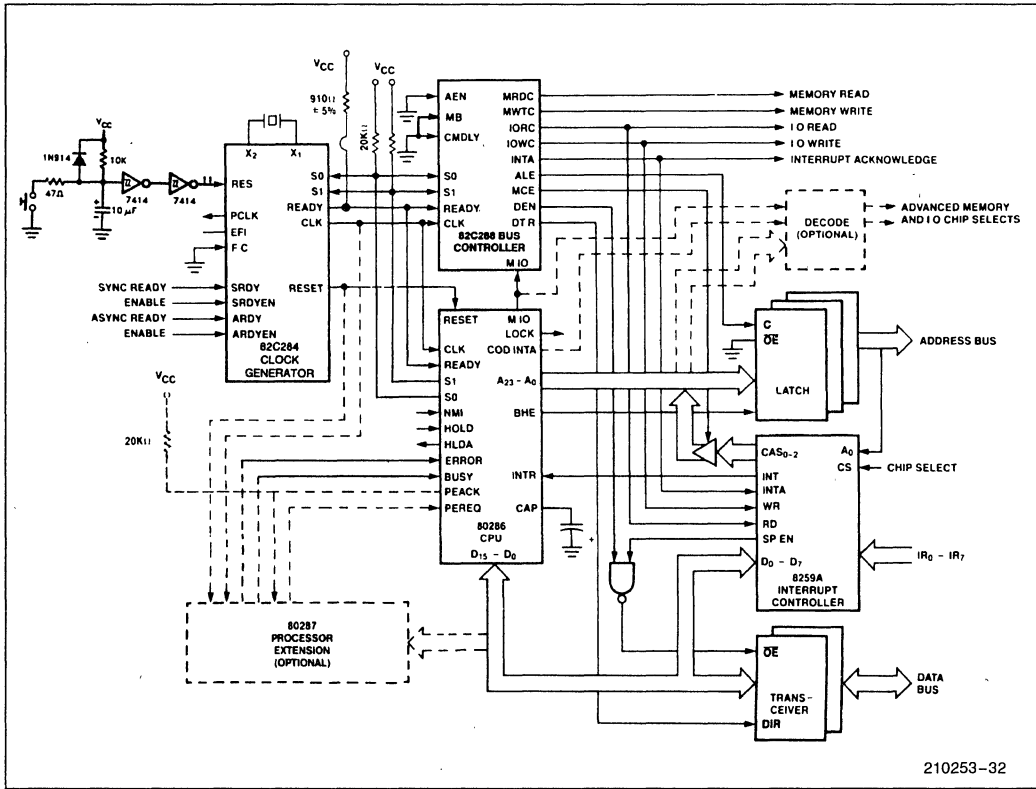


Figure 31. Basic 80286 System Configuration

SYSTEM CONFIGURATIONS

The versatile bus structure of the 80286 microsystem, with a full complement of support chips, allows flexible configuration of a wide range of systems. The basic configuration, shown in Figure 31, is similar to an 8086 maximum mode system. It includes the CPU plus an 8259A interrupt controller, 82C284 clock generator, and the 82C288 Bus Controller.

As indicated by the dashed lines in Figure 31, the ability to add processor extensions is an integral feature of 80286 microsystems. The processor extension interface allows external hardware to perform special functions and transfer data concurrent with CPU execution of other instructions. Full system integrity is maintained because the 80286 supervises all data transfers and instruction execution for the processor extension.

The 80287 has all the instructions and data types of an 8087. The 80287 NPX can perform numeric calculations and data transfers concurrently with CPU program execution. Numerics code and data have the same integrity as all other information protected by the 80286 protection mechanism.

The 80286 can overlap chip select decoding and address propagation during the data transfer for the previous bus operation. This information is latched by ALE during the middle of a T_S cycle. The latched chip select and address information remains stable during the bus operation while the next cycle's address is being decoded and propagated into the system. Decode logic can be implemented with a high speed bipolar PROM.

The optional decode logic shown in Figure 31 takes advantage of the overlap between address and data of the 80286 bus cycle to generate advanced memory and IO-select signals. This minimizes system

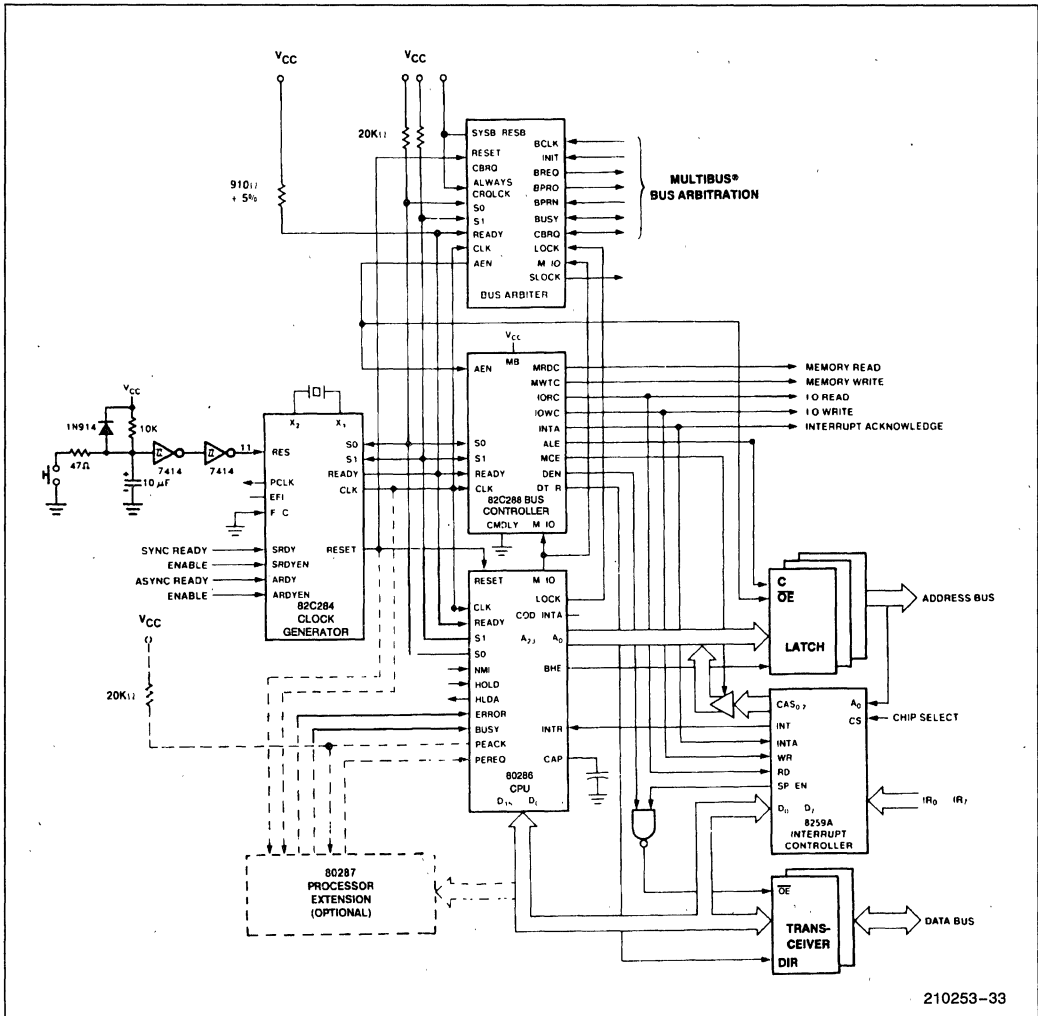


Figure 32. MULTIBUS® System Bus Interface

performance degradation caused by address propagation and decode delays. In addition to selecting memory and I/O, the advanced selects may be used with configurations supporting local and system buses to enable the appropriate bus interface for each bus cycle. The COD/INTA and M/I/O signals are applied to the decode logic to distinguish between interrupt, I/O, code and data bus cycles.

By adding the 82289 bus arbiter chip, the 80286 provides a MULTIBUS system bus interface as shown in Figure 32. The ALE output of the 82C288 for the

MULTIBUS bus is connected to its CMDLY input to delay the start of commands one system CLK as required to meet MULTIBUS address and write data setup times. This arrangement will add at least one extra T_c state to each bus operation which uses the MULTIBUS.

A second 82C288 bus controller and additional latches and transceivers could be added to the local bus of Figure 32. This configuration allows the 80286 to support an on-board bus for local memory and peripherals, and the MULTIBUS for system bus interfacing.

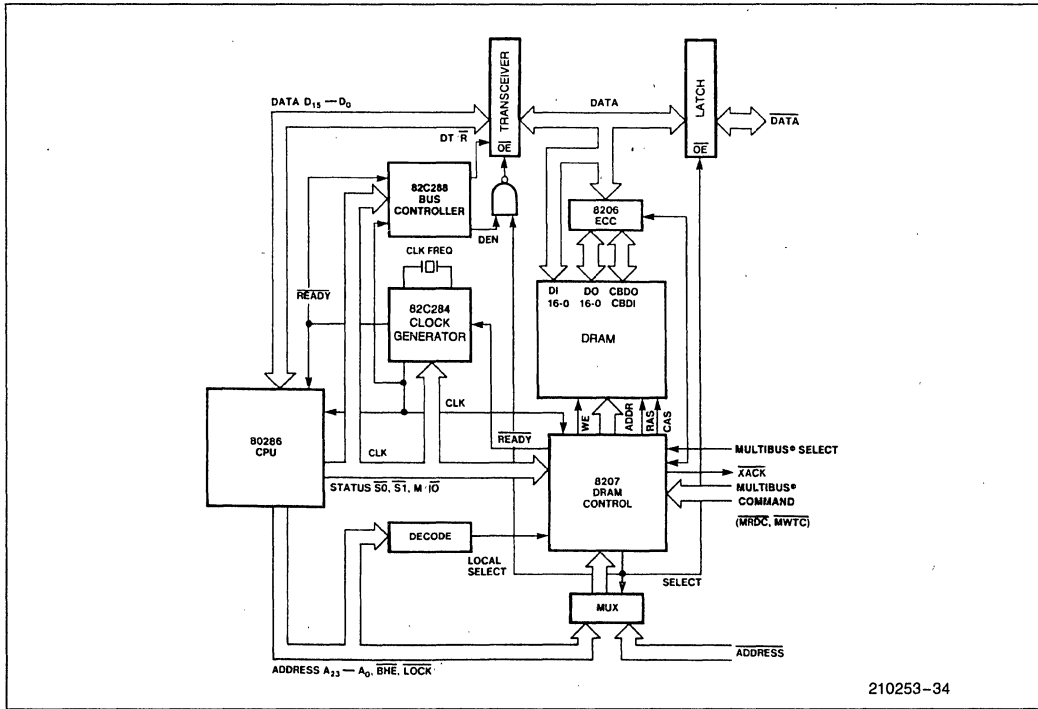


Figure 33. 80286 System Configuration with Dual-Ported Memory

Figure 33 shows the addition of dual ported dynamic memory between the MULTIBUS system bus and the 80286 local bus. The dual port interface is provided by the 8207 Dual Port DRAM Controller. The 8207 runs synchronously with the CPU to maximize throughput for local memory references. It also arbitrates between requests from the local and system buses and performs functions such as refresh,

initialization of RAM, and read/modify/write cycles. The 8207 combined with the 8206 Error Checking and Correction memory controller provide for single bit error correction. The dual-ported memory can be combined with a standard MULTIBUS system bus interface to maximize performance and protection in multiprocessor system configurations.

Table 16. 80286 Systems Recommended Pull Up Resistor Values

80286 Pin and Name	Pullup Value	Purpose
4— $\overline{S1}$	20 K Ω \pm 10%	Pull $\overline{S0}$, $\overline{S1}$, and \overline{PEACK} inactive during 80286 hold periods(1)
5— $\overline{S0}$		
6— \overline{PEACK}		
63— \overline{READY}	910 Ω \pm 5%	Pull \overline{READY} inactive within required minimum time ($C_L = 150$ pF, $I_R \leq 7$ mA)

NOTE:

1. Pull-up resistors are not required on $\overline{S0}$ and $\overline{S1}$ when the corresponding pins of the 82C284 are connected to $\overline{S0}$ and $\overline{S1}$.

I²CTM-286 System Design Considerations

One of the advantages of using the 80286 is that full in-circuit emulation debugging support is provided through the I²C system 80286 probe. To utilize this powerful tool it is necessary that the system designer be aware of a few minor parametric and

functional differences between the 80286 and I²C system 80286 probe. The I²C data sheet (I²C Integrated Instrumentation and In-Circuit Emulation System, order #210469) contains a detailed description of these design considerations. It is recommended that this document be reviewed by the 80286 system designer to determine whether or not these differences affect his design.

PACKAGE THERMAL SPECIFICATIONS

The 80286 Microprocessor is specified for operation when case temperature (T_C) is within the range 0°C–85°C. Case temperature, unlike ambient temperature, is easily measured in any environment to determine whether the 80286 Microprocessor is within the specified operating range. The case temperature should be measured at the center of the top surface of the component.

The maximum ambient temperature (T_A) allowable without violating T_C specifications can be calculated from the equations shown below. T_J is the 80286 junction temperature. P is the power dissipated by the 80286.

$$T_J = T_C + P * \theta_{JC}$$

$$T_A = T_J + P * \theta_{JA}$$

$$T_C = T_A + P * [\theta_{JA} - \theta_{JC}]$$

Values for θ_{JA} and θ_{JC} are given in Table 17. θ_{JA} is given at various airflows. Table 18 shows the maximum T_A allowable (without exceeding T_C) at various airflows. Note that the 80286 PLCC package has an internal heat spreader. T_A can be further improved by attaching "fins" or an external "heat sink" to the package.

Junction temperature calculations should use an I_{CC} value that is measured without external resistive loads. The external resistive loads dissipate additional power external to the 80286 and not on the die. This increases the resistor temperature, not the die temperature. The full capacitive load ($C_L = 100$ pF) should be applied during the I_{CC} measurement.

Table 17. Thermal Resistances (°C/Watt) θ_{JC} and θ_{JA}

Package	θ_{JC}	θ_{JA} versus Airflow — ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
68-Lead LCC	8	28	22	16	13	12	11
68-Lead PGA	5.5	28	22	16	15	14	13
68-Lead PLCC w/ Internal Heat Spreader	8	28	23	21	18	16	15

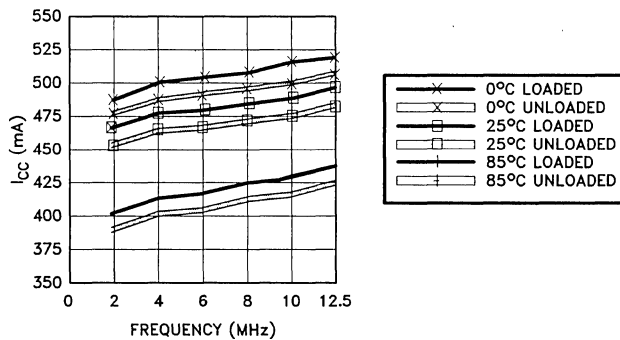
Table 18. Maximum T_A at Various Airflows

Package	T_A (°C) versus Airflow — ft/min (m/sec)					
	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
68-Lead LCC	35	50	65	72	75	77
68-Lead PGA	29	44	59	62	64	66
68-Lead PLCC w/ Internal Heat Spreader	35	48	53	60	65	68

NOTE:

The numbers in Table 18 were calculated using an I_{CC} of 450 mA, which is representative of the worst case I_{CC} at $T_C = 85^\circ\text{C}$ with the outputs unloaded.

Typical I_{CC} vs Frequency for Different Output Loads and Case Temperatures



210253-51

NOTES:

1. $V_{CC} = 5.0V$
2. Loaded: $I_{OL} = 2.0\text{ mA}$, $I_{OH} = -400\text{ }\mu\text{A}$, $C_L = 100\text{ pF}$.
 Unloaded: $C_L = 100\text{ pF}$.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias 0°C to +70°C
 Storage Temperature -65°C to +150°C
 Voltage on Any Pin with
 Respect to Ground -1.0V to +7V
 Power Dissipation 3.3W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS ($V_{CC} = 5V \pm 5\%$, $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$)*

Symbol	Parameter	Min	Max	Unit	Test Condition
I_{CC}	Supply Current (0°C Turn On)		600	mA	(Note 1)
C_{CLK}	CLK Input Capacitance		20	pF	(Note 2)
C_{IN}	Other Input Capacitance		10	pF	(Note 2)
C_O	Input/Output Capacitance		20	pF	(Note 2)

NOTES:

1. Tested at worst case load and maximum frequency.
2. These are not tested. They are guaranteed by design characterization.

D.C. CHARACTERISTICS

($V_{CC} = 5V \pm 5\%$, $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$)* Tested at the minimum operating frequency of the part.

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{IL}	Input LOW Voltage	-0.5	0.8	V	
V_{IH}	Input HIGH Voltage	2.0	$V_{CC} + 0.5$	V	
V_{ILC}	CLK Input LOW Voltage	-0.5	0.6	V	
V_{IHC}	CLK Input HIGH Voltage	3.8	$V_{CC} + 0.5$	V	
V_{OL}	Output LOW Voltage		0.45	V	$I_{OL} = 2.0$ mA
V_{OH}	Output HIGH Voltage	2.4		V	$I_{OH} = -400$ μ A
I_{LI}	Input Leakage Current		± 10	μ A	$0V \leq V_{IN} \leq V_{CC}$
I_{LCR}	Input CLK, RESET Leakage Current		± 10	μ A	$0.45 \leq V_{IN} \leq V_{CC}$
I_{LCR}	Input CLK, RESET Leakage Current		± 1	mA	$0 \leq V_{IN} < 0.45$
I_{IL}	Input Sustaining Current on BUSY and ERROR Pins	-30	-500	μ A	$V_{IN} = 0V$
I_{LO}	Output Leakage Current		± 10	μ A	$0.45 \leq V_{OUT} \leq V_{CC}$ $25^{\circ}C \leq T_{CASE} \leq 85^{\circ}C$
I_{LO}	Output Leakage Current		± 20	μ A	$0.45V \leq V_{OUT} \leq V_{CC}$ $0^{\circ}C \leq T_{CASE} \leq 25^{\circ}C$
I_{LO}	Output Leakage Current		± 1	mA	$0 \leq V_{OUT} < 0.45$

* T_A is guaranteed from 0°C to +55°C as long as T_{CASE} is not exceeded.

A.C. CHARACTERISTICS ($V_{CC} = 5V \pm 5\%$, $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$)*

AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in datasheet waveforms, unless otherwise noted.

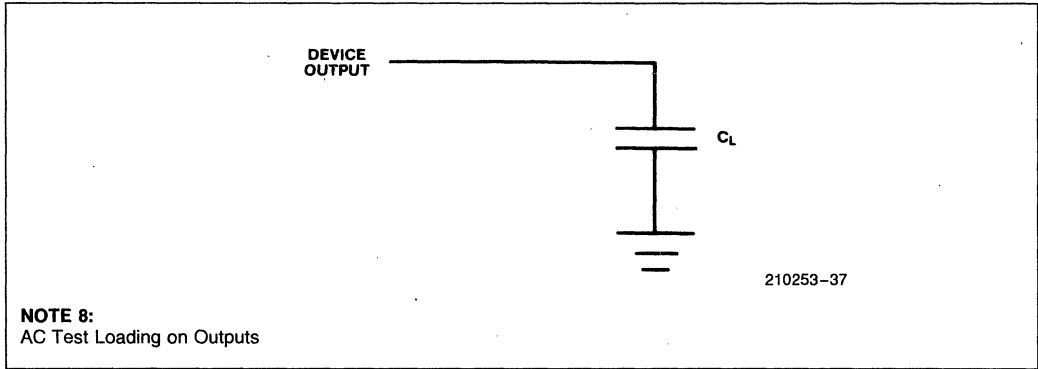
Symbol	Parameter	8 MHz		10 MHz		12.5 MHz		Unit	Test Condition
		-8 Min	-8 Max	-10 Min	-10 Max	-12 Min	-12 Max		
1	System Clock (CLK) Period	62	250	50	250	40	250	ns	
2	System Clock (CLK) LOW Time	15	225	12	232	11	237	ns	at 1.0V
3	System Clock (CLK) HIGH Time	25	235	16	239	13	239	ns	at 3.6V
17	System Clock (CLK) Rise Time		10		8	—	8	ns	1.0V to 3.6V, (Note 7)
18	System Clock (CLK) Fall Time		10		8	—	8	ns	3.6V to 1.0V, (Note 7)
4	Asynch. Inputs Setup Time	20		20		15		ns	(Note 1)
5	Asynch. Inputs Hold Time	20		20		15		ns	(Note 1)
6	RESET Setup Time	28		23		18		ns	
7	RESET Hold Time	5		5		5		ns	
8	Read Data Setup Time	10		8		5		ns	
9	Read Data Hold Time	8		8		6		ns	
10	\overline{READY} Setup Time	38		26		22		ns	
11	\overline{READY} Hold Time	25		25		20		ns	
12	Status/ \overline{PEACK} Valid Delay	1	40	—	—	—	—	ns	(Notes 2, 3)
12a1	Status Active Delay	—	—	1	22	3	18	ns	(Notes 2, 3)
12a2	\overline{PEACK} Active Delay	—	—	1	22	3	20	ns	(Notes 2, 3)
12b	Status/ \overline{PEACK} Inactive Delay	—	—	1	30	3	22	ns	(Notes 2, 3)
13	Address Valid Delay	1	60	1	35	1	32	ns	(Notes 2, 3)
14	Write Data Valid Delay	0	50	0	30	0	30	ns	(Notes 2, 3)
15	Address/Status/Data Float Delay	0	50	0	47	0	32	ns	(Notes 2, 4, 7)
16	HLDA Valid Delay	0	50	0	47	0	27	ns	(Notes 2, 3)
19	Address Valid To Status Valid Setup Time	38		27		22		ns	(Notes 3, 5, 6, 7)

* T_A is guaranteed from $0^{\circ}C$ to $+55^{\circ}C$ as long as T_{CASE} is not exceeded.

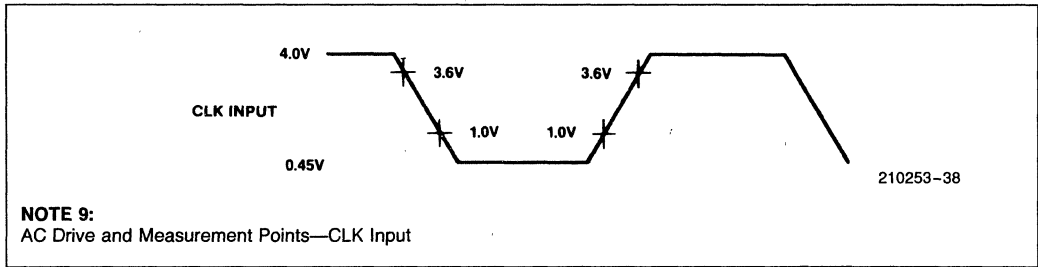
NOTES:

- Asynchronous inputs are INTR, NMI, HOLD, PAREQ, ERROR, and BUSY. This specification is given only for testing purposes, to assure recognition at a specific CLK edge.
- Delay from 1.0V on the CLK, to 0.8V or 2.0V or float on the output as appropriate for valid or floating condition.
- Output load: $C_L = 100$ pF.
- Float condition occurs when output current is less than I_{LO} in magnitude.
- Delay measured from-address either reaching 0.8V or 2.0V (valid) to status going active reaching 2.0V or status going inactive reaching 0.8V.
- For load capacitance of 10 pF or more on STATUS/ \overline{PEACK} lines, subtract typically 7 ns.
- These are not tested. They are guaranteed by design characterization.

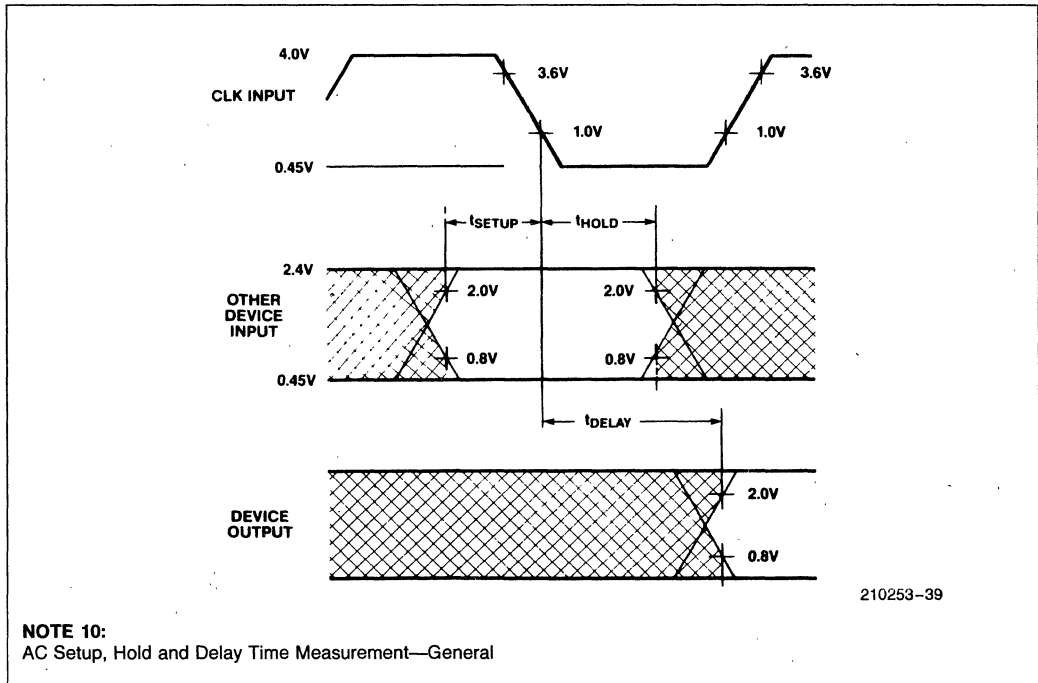
A.C. CHARACTERISTICS (Continued)



NOTE 8:
AC Test Loading on Outputs



NOTE 9:
AC Drive and Measurement Points—CLK Input



NOTE 10:
AC Setup, Hold and Delay Time Measurement—General

A.C. CHARACTERISTICS (Continued)

82C284 Timing Requirements

Symbol	Parameter	82C284-8		82C284-10		82C284-12		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
11	SRDY/SRDYEN Setup Time	20		18		18		ns	
12	SRDY/SRDYEN Hold Time	0		2		2		ns	
13	ARDY/ARDYEN Setup Time	0		0		0		ns	(Note 1)
14	ARDY/ARDYEN Hold Time	30		30		25		ns	(Note 1)
19	PCLK Delay	0	45	0	35	0	23	ns	C _L = 75 pF I _{OL} = 5 mA I _{OH} = -1 mA

NOTE 1:

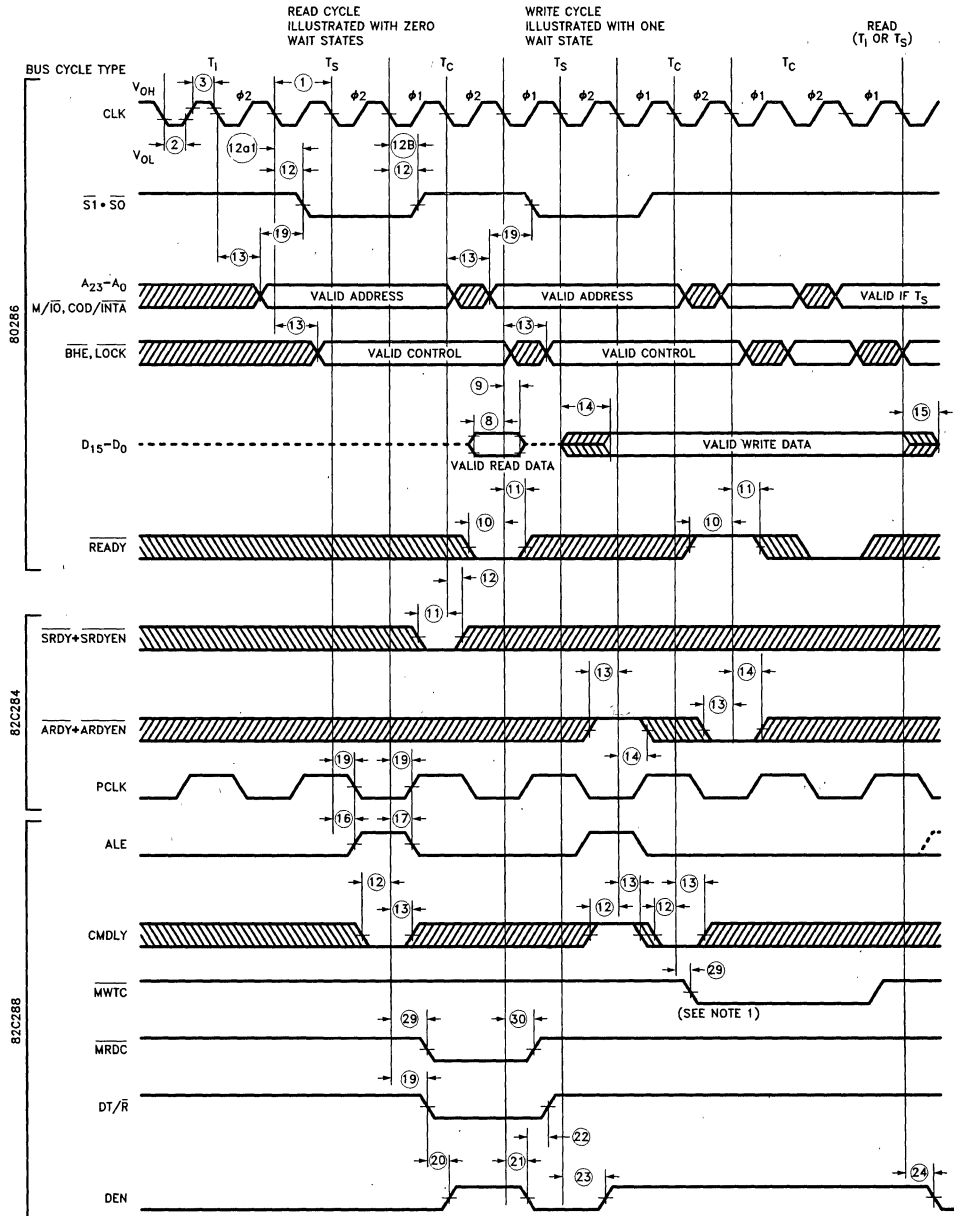
These times are given for testing purposes to assure a predetermined action.

82C288 Timing Requirements

Symbol	Parameter		82C288-8		82C288-10		82C288-12		Units	Test Conditions
			Min	Max	Min	Max	Min	Max		
12	CMDLY Setup Time		20		15		15		ns	
13	CMDLY Hold Time		1		1		1		ns	
30	Command Delay from CLK	Command Inactive	5	20	5	20	5	20	ns	C _L = 300 pF max I _{OL} = 32 mA max I _{OH} = -5 mA max
29		Command Active	3	25	3	21	3	21		
16	ALE Active Delay		3	20	3	16	3	16	ns	C _L = 150 pF I _{OL} = 16 mA max I _{OH} = -1 mA max
17	ALE Inactive Delay			25		19		19	ns	
19	DT/ \bar{R} Read Active Delay			25		23		23	ns	
22	DT/ \bar{R} Read Inactive Delay		5	35	5	20	5	18	ns	
20	DEN Read Active Delay		5	35	5	21	5	21	ns	
21	DEN Read Inactive Delay		3	35	3	21	3	19	ns	
23	DEN Write Active Delay			30		23		23	ns	
24	DEN Write Inactive Delay		3	30	3	19	3	19	ns	

WAVEFORMS

MAJOR CYCLE TIMING

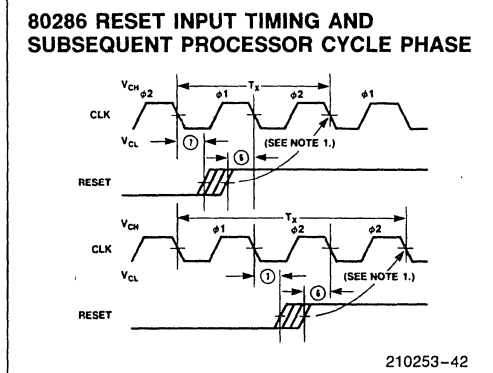
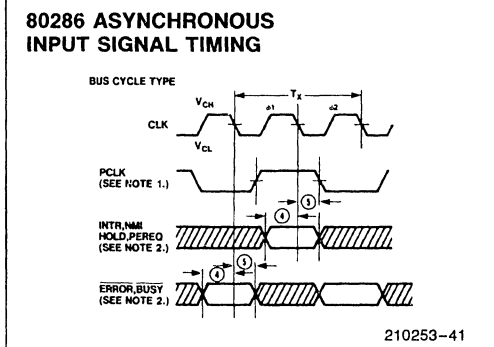


210253-40

NOTE:

1. The modified timing is due to the $\overline{\text{CMDLY}}$ signal being active.

WAVEFORMS (Continued)

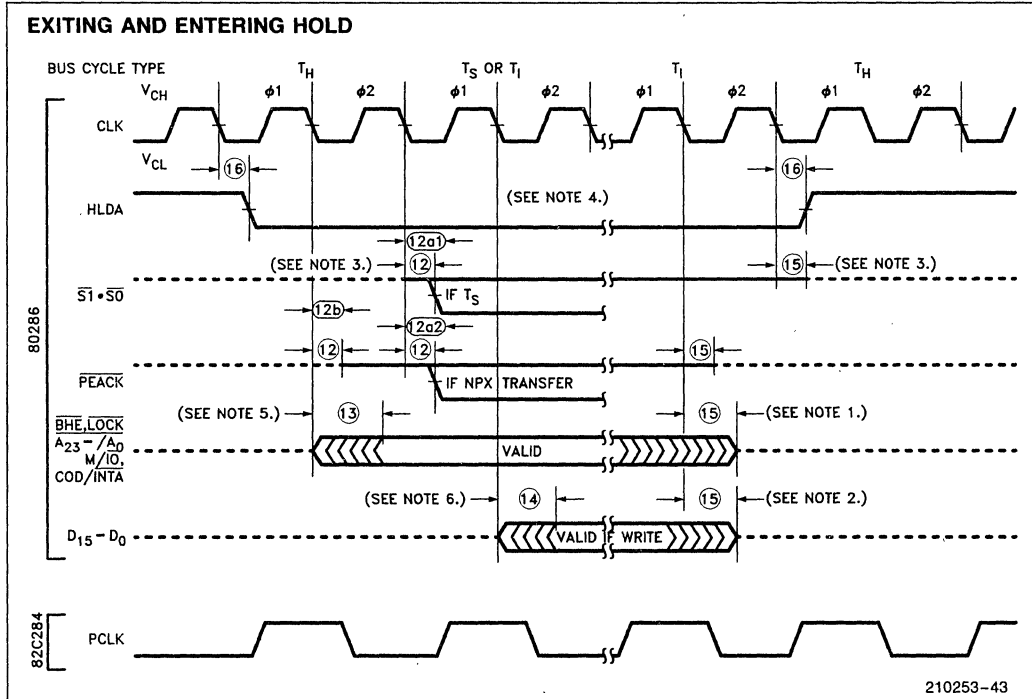


NOTES:

1. PCLK indicates which processor cycle phase will occur on the next CLK. PCLK may not indicate the correct phase until the first bus cycle is performed.
2. These inputs are asynchronous. The setup and hold times shown assure recognition for testing purposes.

NOTE:

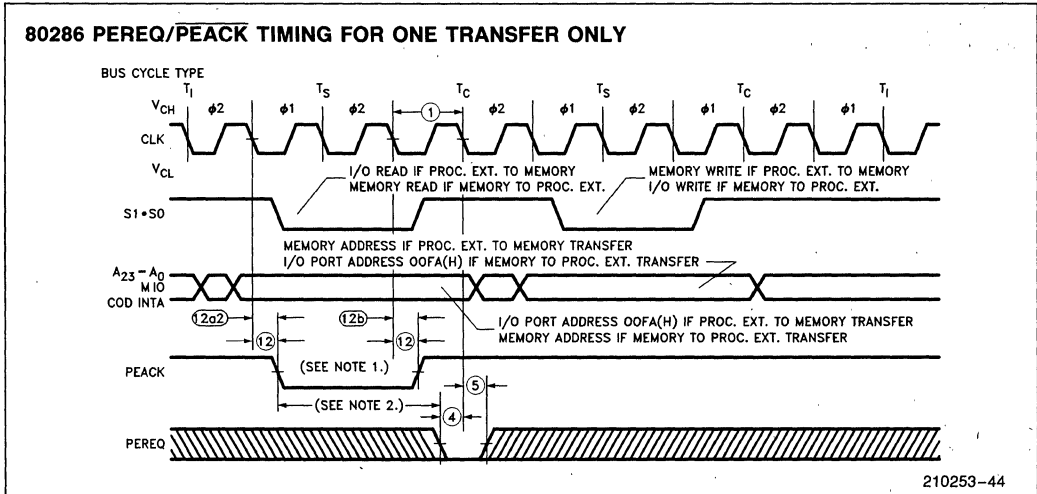
When RESET meets the setup time shown, the next CLK will start or repeat $\phi 2$ of a processor cycle.



NOTES:

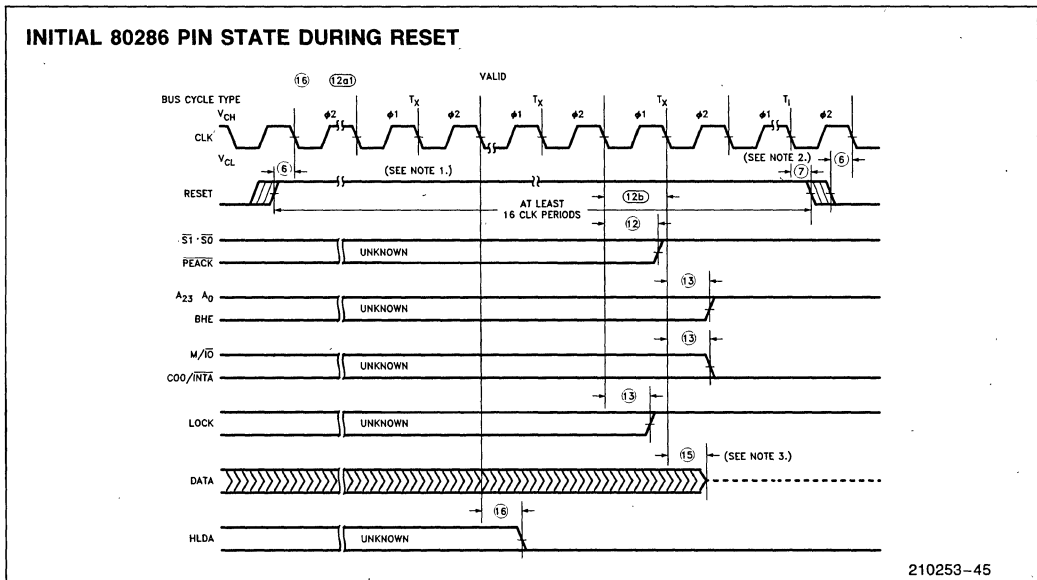
1. These signals may not be driven by the 80286 during the time shown. The worst case in terms of latest float time is shown.
2. The data bus will be driven as shown if the last cycle before T_I in the diagram was a write T_C .
3. The 80286 floats its status pins during T_H . External $20\text{ K}\Omega$ resistors keep these signals high (see Table 16).
4. For HOLD request set up to HLDA, refer to Figure 29.
5. $\overline{\text{BHE}}$ and $\overline{\text{LOCK}}$ are driven at this time but will not become valid until T_S .
6. The data bus will remain in 3-state OFF if a read cycle is performed.

WAVEFORMS (Continued)



NOTES:

1. PEACK always goes active during the first bus operation of a processor extension data operand transfer sequence. The first bus operation will be either a memory read at operand address or I/O read at port address OOFA(H).
2. To prevent a second processor extension data operand transfer, the worst case maximum time (Shown above) is: $3 \times \textcircled{0} - 12a_{2\text{max}} - \textcircled{0}_{\text{min}}$. The actual, configuration dependent, maximum time is: $3 \times \textcircled{0} - 12a_{2\text{max}} - \textcircled{0}_{\text{min}} + A \times 2 \times \textcircled{0}$. A is the number of extra T_C states added to either the first or second bus operation of the processor extension data operand transfer sequence.



NOTES:

1. Setup time for RESET ↑ may be violated with the consideration that φ1 of the processor clock may begin one system CLK period later.
2. Setup and hold times for RESET ↓ must be met for proper operation, but RESET ↓ may occur during φ1 or φ2.
3. The data bus is only guaranteed to be in 3-state OFF at the time shown.

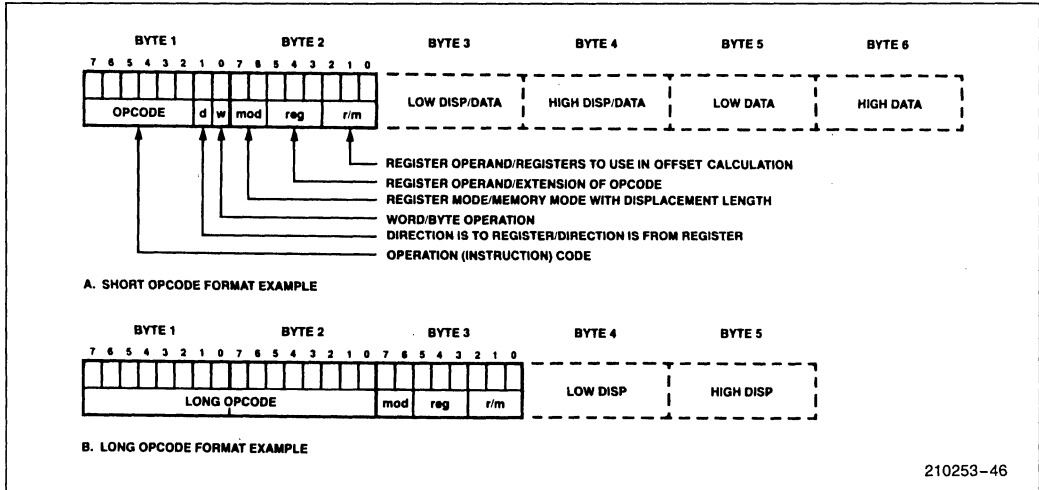


Figure 35. 80286 Instruction Format Examples

80286 INSTRUCTION SET SUMMARY

Instruction Timing Notes

The instruction clock counts listed below establish the maximum execution rate of the 80286. With no delays in bus cycles, the actual clock count of an 80286 program will average 5% more than the calculated clock count, due to instruction sequences which execute faster than they can be fetched from memory.

To calculate elapsed times for instruction sequences, multiply the sum of all instruction clock counts, as listed in the table below, by the processor clock period. An 8 MHz processor clock has a clock period of 125 nanoseconds and requires an 80286 system clock (CLK input) of 16 MHz.

Instruction Clock Count Assumptions

1. The instruction has been prefetched, decoded, and is ready for execution. Control transfer instruction clock counts include all time required to fetch, decode, and prepare the next instruction for execution.
2. Bus cycles do not require wait states.
3. There are no processor extension data transfer or local bus HOLD requests.
4. No exceptions occur during instruction execution.

Instruction Set Summary Notes

Addressing displacements selected by the MOD field are not shown. If necessary they appear after the instruction fields shown.

Above/below refers to unsigned value

Greater refers to positive signed value

Less refers to less positive (more negative) signed values

if d = 1 then to register; if d = 0 then from register

if w = 1 then word instruction; if w = 0 then byte instruction

if s = 0 then 16-bit immediate data form the operand

if s = 1 then an immediate data byte is sign-extended to form the 16-bit operand

x don't care

z used for string primitives for comparison with ZF FLAG

If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand

* = add one clock if offset calculation requires summing 3 elements

n = number of times repeated

m = number of bytes of code in next instruction

Level (L)—Lexical nesting level of the procedure

The following comments describe possible exceptions, side effects, and allowed usage for instructions in both operating modes of the 80286.

REAL ADDRESS MODE ONLY

1. This is a protected mode instruction. Attempted execution in real address mode will result in an undefined opcode exception (6).
2. A segment overrun exception (13) will occur if a word operand reference at offset FFFF(H) is attempted.
3. This instruction may be executed in real address mode to initialize the CPU for protected mode.
4. The IOPL and NT fields will remain 0.
5. Processor extension segment overrun interrupt (9) will occur if the operand exceeds the segment limit.

EITHER MODE

6. An exception may occur, depending on the value of the operand.
7. LOCK is automatically asserted regardless of the presence or absence of the LOCK instruction prefix.
8. LOCK does not remain active between all operand transfers.

PROTECTED VIRTUAL ADDRESS MODE ONLY

9. A general protection exception (13) will occur if the memory operand cannot be used due to either a segment limit or access rights violation. If a stack segment limit is violated, a stack segment overrun exception (12) occurs.
10. For segment load operations, the CPL, RPL, and DPL must agree with privilege rules to avoid an exception. The segment must be present to avoid a not-present exception (11). If the SS register is the destination, and a segment not-present violation occurs, a stack exception (12) occurs.

11. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert LOCK to maintain descriptor integrity in multiprocessor systems.
12. JMP, CALL, INT, RET, IRET instructions referring to another code segment will cause a general protection exception (13) if any privilege rule is violated.
13. A general protection exception (13) occurs if $CPL \neq 0$.
14. A general protection exception (13) occurs if $CPL > IOPL$.
15. The IF field of the flag word is not updated if $CPL > IOPL$. The IOPL field is updated only if $CPL = 0$.
16. Any violation of privilege rules as applied to the selector operand do not cause a protection exception; rather, the instruction does not return a result and the zero flag is cleared.
17. If the starting address of the memory operand violates a segment limit, or an invalid access is attempted, a general protection exception (13) will occur before the ESC instruction is executed. A stack segment overrun exception (12) will occur if the stack limit is violated by the operand's starting address. If a segment limit is violated during an attempted data transfer then a processor extension segment overrun exception (9) occurs.
18. The destination of an INT, JMP, CALL, RET or IRET instruction must be in the defined limit of a code segment or a general protection exception (13) will occur.

80286 INSTRUCTION SET SUMMARY

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
DATA TRANSFER					
MOV = Move:					
Register to Register/Memory	1 0 0 0 1 0 0 w mod reg r/m	2,3*	2,3*	2	9
Register/memory to register	1 0 0 0 1 0 1 w mod reg r/m	2,5*	2,5*	2	9
Immediate to register/memory	1 1 0 0 0 1 1 w mod 0 0 0 r/m data data if w = 1	2,3*	2,3*	2	9
Immediate to register	1 0 1 1 w reg data data if w = 1	2	2		
Memory to accumulator	1 0 1 0 0 0 0 w addr-low addr-high	5	5	2	9
Accumulator to memory	1 0 1 0 0 0 1 w addr-low addr-high	3	3	2	9
Register/memory to segment register	1 0 0 0 1 1 1 0 mod 0 reg r/m	2,5*	17,19*	2	9,10,11
Segment register to register/memory	1 0 0 0 1 1 0 0 mod 0 reg r/m	2,3*	2,3*	2	9
PUSH = Push:					
Memory	1 1 1 1 1 1 1 1 mod 1 1 0 r/m	5*	5*	2	9
Register	0 1 0 1 0 reg	3	3	2	9
Segment register	0 0 0 reg 1 1 0	3	3	2	9
Immediate	0 1 1 0 1 0 s 0 data data if s = 0	3	3	2	9
PUSHA = Push All	0 1 1 0 0 0 0 0	17	17	2	9
POP = Pop:					
Memory	1 0 0 0 1 1 1 1 mod 0 0 0 r/m	5*	5*	2	9
Register	0 1 0 1 1 reg	5	5	2	9
Segment register	0 0 0 reg 1 1 1 (reg ≠ 01)	5	20	2	9,10,11
POPA = Pop All	0 1 1 0 0 0 0 1	19	19	2	9
XCHG = Exchange:					
Register/memory with register	1 0 0 0 0 1 1 w mod reg r/m	3,5*	3,5*	2,7	7,9
Register with accumulator	1 0 0 1 0 reg	3	3		
IN = Input from:					
Fixed port	1 1 1 0 0 1 0 w port	5	5		14
Variable port	1 1 1 0 1 1 0 w	5	5		14
OUT = Output to:					
Fixed port	1 1 1 0 0 1 1 w port	3	3		14
Variable port	1 1 1 0 1 1 1 w	3	3		14
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1	5	5		9
LEA = Load EA to register	1 0 0 0 1 1 0 1 mod reg r/m	3*	3*		
LDS = Load pointer to DS	1 1 0 0 0 1 0 1 mod reg r/m (mod ≠ 11)	7*	21*	2	9,10,11
LES = Load pointer to ES	1 1 0 0 0 1 0 0 mod reg r/m (mod ≠ 1)	7*	21*	2	9,10,11

Shaded areas indicate instructions not available in 8086, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
DATA TRANSFER (Continued)					
LAHF Load AH with flags	10011111	2	2		
SAHF = Store AH into flags	10011110	2	2		
PUSHF = Push flags	10011100	3	3	2	9
POPF = Pop flags	10011101	5	5	2,4	9,15
ARITHMETIC					
ADD = Add:					
Reg/memory with register to either	000000d w mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	100000s w mod 000 r/m data data if s w = 01	3,7*	3,7*	2	9
Immediate to accumulator	0000010 w data data if w = 1	3	3		
ADC = Add with carry:					
Reg/memory with register to either	000100d w mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	100000s w mod 010 r/m data data if s w = 01	3,7*	3,7*	2	9
Immediate to accumulator	0001010 w data data if w = 1	3	3		
INC = Increment:					
Register/memory	1111111 w mod 000 r/m	2,7*	2,7*	2	9
Register	01000 reg	2	2		
SUB = Subtract:					
Reg/memory and register to either	001010d w mod reg r/m	2,7*	2,7*	2	9
Immediate from register/memory	100000s w mod 101 r/m data data if s w = 01	3,7*	3,7*	2	9
Immediate from accumulator	0010110 w data data if w = 1	3	3		
SBB = Subtract with borrow:					
Reg/memory and register to either	000110d w mod reg r/m	2,7*	2,7*	2	9
Immediate from register/memory	100000s w mod 011 r/m data data if s w = 01	3,7*	3,7*	2	9
Immediate from accumulator	0001110 w data data if w = 1	3	3		
DEC = Decrement					
Register/memory	1111111 w mod 001 r/m	2,7*	2,7*	2	9
Register	01001 reg	2	2		
CMP = Compare					
Register/memory with register	0011101 w mod reg r/m	2,6*	2,6*	2	9
Register with register/memory	0011100 w mod reg r/m	2,7*	2,7*	2	9
Immediate with register/memory	100000s w mod 111 r/m data data if s w = 01	3,6*	3,6*	2	9
Immediate with accumulator	0011110 w data data if w = 1	3	3		
NEG = Change sign	1111011 w mod 011 r/m	2	7*	2	9
AAA = ASCII adjust for add	00110111	3	3		
DAA = Decimal adjust for add	00100111	3	3		

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS																	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode																
ARITHMETIC (Continued)																					
AAS = ASCII adjust for subtract	0 0 1 1 1 1 1 1	3	3																		
DAS = Decimal adjust for subtract	0 0 1 0 1 1 1 1	3	3																		
MUL = Multiply (unsigned):	1 1 1 1 0 1 1 w mod 1 0 0 r/m																				
Register-Byte		13	13																		
Register-Word		21	21																		
Memory-Byte		16*	16*	2	9																
Memory-Word		24*	24*	2	9																
IMUL = Integer multiply (signed):	1 1 1 1 0 1 1 w mod 1 0 1 r/m																				
Register-Byte		13	13																		
Register-Word		21	21																		
Memory-Byte		16*	16*	2	9																
Memory-Word		24*	24*	2	9																
IMUL = Integer immediate multiply (signed)	0 1 1 0 1 0 s 1 mod reg r/m data data if s = 0	21,24*	21,24*	2	9																
DIV = Divide (unsigned)	1 1 1 1 0 1 1 w mod 1 1 0 r/m																				
Register-Byte		14	14	6	6																
Register-Word		22	22	6	6																
Memory-Byte		17*	17*	2,6	6,9																
Memory-Word		25*	25*	2,6	6,9																
IDIV = Integer divide (signed)	1 1 1 1 0 1 1 w mod 1 1 1 r/m																				
Register-Byte		17	17	6	6																
Register-Word		25	25	6	6																
Memory-Byte		20*	20*	2,6	6,9																
Memory-Word		28*	28*	2,6	6,9																
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0	16	16																		
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1 0 0 0 0 1 0 1 0	14	14																		
CBW = Convert byte to word	1 0 0 1 1 0 0 0	2	2																		
CWD = Convert word to double word	1 0 0 1 1 0 0 1	2	2																		
LOGIC																					
Shift/Rotate Instructions:																					
Register/Memory by 1	1 1 0 1 0 0 0 w mod TTT r/m	2,7*	2,7*	2	9																
Register/Memory by CL	1 1 0 1 0 0 1 w mod TTT r/m	5+n,8+n*	5+n,8+n*	2	9																
Register/Memory by Count	1 1 0 0 0 0 0 w mod TTT r/m count	5+n,8+n*	5+n,8+n*	2	9																
		<table border="0"> <tr> <td>TTT</td> <td>Instruction</td> </tr> <tr> <td>000</td> <td>ROL</td> </tr> <tr> <td>001</td> <td>ROR</td> </tr> <tr> <td>010</td> <td>RCL</td> </tr> <tr> <td>011</td> <td>RCR</td> </tr> <tr> <td>100</td> <td>SHL/SAL</td> </tr> <tr> <td>101</td> <td>SHR</td> </tr> <tr> <td>111</td> <td>SAR</td> </tr> </table>				TTT	Instruction	000	ROL	001	ROR	010	RCL	011	RCR	100	SHL/SAL	101	SHR	111	SAR
TTT	Instruction																				
000	ROL																				
001	ROR																				
010	RCL																				
011	RCR																				
100	SHL/SAL																				
101	SHR																				
111	SAR																				

Shaded areas indicate instructions not available in 8086, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
ARITHMETIC (Continued)					
AND = And:					
Reg/memory and register to either	001000dw mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1000000w mod 100 r/m data data if w=1	3,7*	3,7*	2	9
Immediate to accumulator	0010010w data data if w=1	3	3		
TEST = And function to flags, no result:					
Register/memory and register	1000010w mod reg r/m	2,6*	2,6*	2	9
Immediate data and register/memory	1111011w mod 000 r/m data data if w=1	3,6*	3,6*	2	9
Immediate data and accumulator	1010100w data data if w=1	3	3		
OR = Or:					
Reg/memory and register to either	000010dw mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1000000w mod 001 r/m data data if w=1	3,7*	3,7*	2	9
Immediate to accumulator	0000110w data data if w=1	3	3		
XOR = Exclusive or:					
Reg/memory and register to either	001100dw mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1000000w mod 110 r/m data data if w=1	3,7*	3,7*	2	9
Immediate to accumulator	0011010w data data if w=1	3	3		
NOT = Invert register/memory	1111011w mod 010 r/m	2,7*	2,7*	2	9
STRING MANIPULATION:					
MOVS = Move byte/word	1010010w	5	5	2	9
CMPS = Compare byte/word	1010011w	8	8	2	9
SCAS = Scan byte/word	1010111w	7	7	2	9
LDS = Load byte/wd to AL/AX	1010110w	5	5	2	9
STOS = Stor byte/wd from AL/A	1010101w	3	3	2	9
INS = Input byte/wd from DX port	0110110w	5	5	2	9,14
OUTS = Output byte/wd to DX port	0110111w	5	5	2	9,14
Repeated by count in CX					
MOVsb = Move string	11110011 1010010w	5+4n	5+4n	2	9
CMPSb = Compare string	1111001z 1010011w	5+9n	5+9n	2,8	8,9
SCASb = Scan string	1111001z 1010111w	5+8n	5+8n	2,8	8,9
LDSb = Load string	11110011 1010110w	5+4n	5+4n	2,8	8,9
STOSb = Store string	11110011 1010101w	4+3n	4+3n	2,8	8,9
INSs = Input string	11110011 0110110w	5+4n	5+4n	2	9,14
OUTSs = Output string	11110011 0110111w	5+4n	5+4n	2	9,14

Shaded areas indicate instructions not available in 8086, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS					
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode				
CONTROL TRANSFER									
CALL = Call:									
Direct within segment	<table border="1"><tr><td>1 1 1 0 1 0 0 0</td><td>disp-low</td><td>disp-high</td></tr></table>	1 1 1 0 1 0 0 0	disp-low	disp-high	7 + m	7 + m	2	18	
1 1 1 0 1 0 0 0	disp-low	disp-high							
Register/memory indirect within segment	<table border="1"><tr><td>1 1 1 1 1 1 1 1</td><td>mod 0 1 0</td><td>r/m</td></tr></table>	1 1 1 1 1 1 1 1	mod 0 1 0	r/m	7 + m, 11 + m*	7 + m, 11 + m*	2,8	8,9,18	
1 1 1 1 1 1 1 1	mod 0 1 0	r/m							
Direct intersegment	<table border="1"><tr><td>1 0 0 1 1 0 1 0</td><td>segment offset</td></tr></table>	1 0 0 1 1 0 1 0	segment offset	13 + m	26 + m	2	11,12,18		
1 0 0 1 1 0 1 0	segment offset								
Protected Mode Only (Direct intersegment):	<table border="1"><tr><td>segment selector</td></tr></table>	segment selector							
segment selector									
Via call gate to same privilege level			41 + m		8,11,12,18				
Via call gate to different privilege level, no parameters			82 + m		8,11,12,18				
Via call gate to different privilege level, x parameters			86 + 4x + m		8,11,12,18				
Via TSS			177 + m		8,11,12,18				
Via task gate			182 + m		8,11,12,18				
Indirect intersegment	<table border="1"><tr><td>1 1 1 1 1 1 1 1</td><td>mod 0 1 1</td><td>r/m</td><td>(mod ≠ 11)</td></tr></table>	1 1 1 1 1 1 1 1	mod 0 1 1	r/m	(mod ≠ 11)	16 + m	29 + m*	2	8,9,11,12,18
1 1 1 1 1 1 1 1	mod 0 1 1	r/m	(mod ≠ 11)						
Protected Mode Only (Indirect intersegment):									
Via call gate to same privilege level			44 + m*		8,9,11,12,18				
Via call gate to different privilege level, no parameters			83 + m*		8,9,11,12,18				
Via call gate to different privilege level, x parameters			90 + 4x + m*		8,9,11,12,18				
Via TSS			180 + m*		8,9,11,12,18				
Via task gate			185 + m*		8,9,11,12,18				
JMP = Unconditional jump:									
Short/long	<table border="1"><tr><td>1 1 1 0 1 0 1 1</td><td>disp-low</td></tr></table>	1 1 1 0 1 0 1 1	disp-low	7 + m	7 + m		18		
1 1 1 0 1 0 1 1	disp-low								
Direct within segment	<table border="1"><tr><td>1 1 1 0 1 0 0 1</td><td>disp-low</td><td>disp-high</td></tr></table>	1 1 1 0 1 0 0 1	disp-low	disp-high	7 + m	7 + m		18	
1 1 1 0 1 0 0 1	disp-low	disp-high							
Register/memory indirect within segment	<table border="1"><tr><td>1 1 1 1 1 1 1 1</td><td>mod 1 0 0</td><td>r/m</td></tr></table>	1 1 1 1 1 1 1 1	mod 1 0 0	r/m	7 + m, 11 + m*	7 + m, 11 + m*	2	9,18	
1 1 1 1 1 1 1 1	mod 1 0 0	r/m							
Direct intersegment	<table border="1"><tr><td>1 1 1 0 1 0 1 0</td><td>segment offset</td></tr></table>	1 1 1 0 1 0 1 0	segment offset	11 + m	23 + m		11,12,18		
1 1 1 0 1 0 1 0	segment offset								
Protected Mode Only (Direct intersegment):	<table border="1"><tr><td>segment selector</td></tr></table>	segment selector							
segment selector									
Via call gate to same privilege level			38 + m		8,11,12,18				
Via TSS			175 + m		8,11,12,18				
Via task gate			180 + m		8,11,12,18				
Indirect intersegment	<table border="1"><tr><td>1 1 1 1 1 1 1 1</td><td>mod 1 0 1</td><td>r/m</td><td>(mod ≠ 11)</td></tr></table>	1 1 1 1 1 1 1 1	mod 1 0 1	r/m	(mod ≠ 11)	15 + m*	26 + m*	2	8,9,11,12,18
1 1 1 1 1 1 1 1	mod 1 0 1	r/m	(mod ≠ 11)						
Protected Mode Only (Indirect intersegment):									
Via call gate to same privilege level			41 + m*		8,9,11,12,18				
Via TSS			178 + m*		8,9,11,12,18				
Via task gate			183 + m*		8,9,11,12,18				
RET = Return from CALL:									
Within segment	<table border="1"><tr><td>1 1 0 0 0 0 1 1</td></tr></table>	1 1 0 0 0 0 1 1	11 + m	11 + m	2	8,9,18			
1 1 0 0 0 0 1 1									
Within seg adding immed to SP	<table border="1"><tr><td>1 1 0 0 0 0 1 0</td><td>data-low</td><td>data-high</td></tr></table>	1 1 0 0 0 0 1 0	data-low	data-high	11 + m	11 + m	2	8,9,18	
1 1 0 0 0 0 1 0	data-low	data-high							
Intersegment	<table border="1"><tr><td>1 1 0 0 1 0 1 1</td></tr></table>	1 1 0 0 1 0 1 1	15 + m	25 + m	2	8,9,11,12,18			
1 1 0 0 1 0 1 1									
Intersegment adding immediate to SP	<table border="1"><tr><td>1 1 0 0 1 0 1 0</td><td>data-low</td><td>data-high</td></tr></table>	1 1 0 0 1 0 1 0	data-low	data-high	15 + m		2	8,9,11,12,18	
1 1 0 0 1 0 1 0	data-low	data-high							
Protected Mode Only (RET):									
To different privilege level			55 + m		9,11,12,18				

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS					
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode				
CONTROL TRANSFER (Continued)									
JE/JZ = Jump on equal zero	<table border="1"><tr><td>0 1 1 1 0 1 0 0</td><td>disp</td></tr></table>	0 1 1 1 0 1 0 0	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 0 1 0 0	disp								
JL/JNGE = Jump on less/not greater or equal	<table border="1"><tr><td>0 1 1 1 1 1 0 0</td><td>disp</td></tr></table>	0 1 1 1 1 1 0 0	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 1 1 0 0	disp								
JLE/JNG = Jump on less or equal/not greater	<table border="1"><tr><td>0 1 1 1 1 1 1 0</td><td>disp</td></tr></table>	0 1 1 1 1 1 1 0	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 1 1 1 0	disp								
JB/JNAE = Jump on below/not above or equal	<table border="1"><tr><td>0 1 1 1 0 0 1 0</td><td>disp</td></tr></table>	0 1 1 1 0 0 1 0	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 0 0 1 0	disp								
JBE/JNA = Jump on below or equal/not above	<table border="1"><tr><td>0 1 1 1 0 1 1 0</td><td>disp</td></tr></table>	0 1 1 1 0 1 1 0	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 0 1 1 0	disp								
JP/JPE = Jump on parity/parity even	<table border="1"><tr><td>0 1 1 1 1 0 1 0</td><td>disp</td></tr></table>	0 1 1 1 1 0 1 0	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 1 0 1 0	disp								
JO = Jump on overflow	<table border="1"><tr><td>0 1 1 1 0 0 0 0</td><td>disp</td></tr></table>	0 1 1 1 0 0 0 0	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 0 0 0 0	disp								
JS = Jump on sign	<table border="1"><tr><td>0 1 1 1 1 0 0 0</td><td>disp</td></tr></table>	0 1 1 1 1 0 0 0	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 1 0 0 0	disp								
JNE/JNZ = Jump on not equal/not zero	<table border="1"><tr><td>0 1 1 1 0 1 0 1</td><td>disp</td></tr></table>	0 1 1 1 0 1 0 1	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 0 1 0 1	disp								
JNL/JGE = Jump on not less/greater or equal	<table border="1"><tr><td>0 1 1 1 1 1 0 1</td><td>disp</td></tr></table>	0 1 1 1 1 1 0 1	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 1 1 0 1	disp								
JNLE/JG = Jump on not less or equal/greater	<table border="1"><tr><td>0 1 1 1 1 1 1 1</td><td>disp</td></tr></table>	0 1 1 1 1 1 1 1	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 1 1 1 1	disp								
JNB/JAE = Jump on not below/above or equal	<table border="1"><tr><td>0 1 1 1 0 0 1 1</td><td>disp</td></tr></table>	0 1 1 1 0 0 1 1	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 0 0 1 1	disp								
JNBE/JA = Jump on not below or equal/above	<table border="1"><tr><td>0 1 1 1 0 1 1 1</td><td>disp</td></tr></table>	0 1 1 1 0 1 1 1	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 0 1 1 1	disp								
JNP/JPO = Jump on not par/par odd	<table border="1"><tr><td>0 1 1 1 1 0 1 1</td><td>disp</td></tr></table>	0 1 1 1 1 0 1 1	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 1 0 1 1	disp								
JNO = Jump on not overflow	<table border="1"><tr><td>0 1 1 1 0 0 0 1</td><td>disp</td></tr></table>	0 1 1 1 0 0 0 1	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 0 0 0 1	disp								
JNS = Jump on not sign	<table border="1"><tr><td>0 1 1 1 1 0 0 1</td><td>disp</td></tr></table>	0 1 1 1 1 0 0 1	disp	7 + m or 3	7 + m or 3		18		
0 1 1 1 1 0 0 1	disp								
LOOP = Loop CX times	<table border="1"><tr><td>1 1 1 0 0 0 1 0</td><td>disp</td></tr></table>	1 1 1 0 0 0 1 0	disp	8 + m or 4	8 + m or 4		18		
1 1 1 0 0 0 1 0	disp								
LOOPZ/LOOPE = Loop while zero/equal	<table border="1"><tr><td>1 1 1 0 0 0 0 1</td><td>disp</td></tr></table>	1 1 1 0 0 0 0 1	disp	8 + m or 4	8 + m or 4		18		
1 1 1 0 0 0 0 1	disp								
LOOPNZ/LOOPNE = Loop while not zero/equal	<table border="1"><tr><td>1 1 1 0 0 0 0 0</td><td>disp</td></tr></table>	1 1 1 0 0 0 0 0	disp	8 + m or 4	8 + m or 4		18		
1 1 1 0 0 0 0 0	disp								
JCXZ = Jump on CX zero	<table border="1"><tr><td>1 1 1 0 0 0 1 1</td><td>disp</td></tr></table>	1 1 1 0 0 0 1 1	disp	8 + m or 4	8 + m or 4		18		
1 1 1 0 0 0 1 1	disp								
ENTER = Enter Procedure	<table border="1"><tr><td>1 1 0 0 1 0 0 0</td><td>data-low</td><td>data-high</td><td>L</td></tr></table>	1 1 0 0 1 0 0 0	data-low	data-high	L			2,8	8,9
1 1 0 0 1 0 0 0	data-low	data-high	L						
L = 0		11	11						
L = 1		15	15	2,8	8,9				
L > 1		16 + 4(L - 1)	16 + 4(L - 1)	2,8	8,9				
LEAVE = Leave Procedure	<table border="1"><tr><td>1 1 0 0 1 0 0 1</td></tr></table>	1 1 0 0 1 0 0 1	5	5					
1 1 0 0 1 0 0 1									
INT = Interrupt:									
Type specified	<table border="1"><tr><td>1 1 0 0 1 1 0 1</td><td>type</td></tr></table>	1 1 0 0 1 1 0 1	type	23 + m		2,7,8			
1 1 0 0 1 1 0 1	type								
Type 3	<table border="1"><tr><td>1 1 0 0 1 1 0 0</td></tr></table>	1 1 0 0 1 1 0 0	23 + m		2,7,8				
1 1 0 0 1 1 0 0									
INTO = Interrupt on overflow	<table border="1"><tr><td>1 1 0 0 1 1 1 0</td></tr></table>	1 1 0 0 1 1 1 0	24 + m or 3 (3 if no interrupt)	(3 if no interrupt)	2,6,8				
1 1 0 0 1 1 1 0									

Shaded areas indicate instructions not available in 8086, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
CONTROL TRANSFER (Continued)					
Protected Mode Only:					
Via interrupt or trap gate to same privilege level			40 + m		7,8,11,12,18
Via interrupt or trap gate to fit different privilege level			78 + m		7,8,11,12,18
Via Task Gate			167 + m		7,8,11,12,18
IRET = Interrupt return	11001111	17 + m	31 + m	2,4	8,9,11,12,15,18
Protected Mode Only:					
To different privilege level			55 + m		8,9,11,12,15,18
To different task (NT = 1)			169 + m		8,9,11,12,18
BOUND = Detect value out of range	01100010 mod reg r/m	13*	13* (Use INT clock count if exception 5)	2,6	6,8,9,11,12,18
PROCESSOR CONTROL					
CLC = Clear carry	11111000	2	2		
CMC = Complement carry	11110101	2	2		
STC = Set carry	11111001	2	2		
CLD = Clear direction	11111100	2	2		
STD = Set direction	11111101	2	2		
CLI = Clear interrupt	11111010	3	3		14
STI = Set interrupt	11111011	2	2		14
HLT = Halt	11110100	2	2		13
WAIT = Wait	10011011	3	3		
LOCK = Bus lock prefix	11110000	0	0		14
CTS = Clear task switched flag	00001111 00000110	2	2	3	13
ESC = Processor Extension Escape	11011111 mod LLL r/m (TTT LLL are opcode to processor extension)	9-20*	9-20*	5,8	8,17
SEG = Segment Override Prefix	001 reg 110	0	0		
PROTECTION CONTROL					
LGDT = Load global descriptor table register	00001111 00000001 mod 010 r/m	11*	11*	2,3	9,13
SGDT = Store global descriptor table register	00001111 00000001 mod 000 r/m	11*	11*	2,3	9
LIDT = Load interrupt descriptor table register	00001111 00000001 mod 011 r/m	12*	12*	2,3	9,13
SIDT = Store interrupt descriptor table register	00001111 00000001 mod 001 r/m	12*	12*	2,3	9
LLDT = Load local descriptor table register from register memory	00001111 00000000 mod 010 r/m		17,19*	1	9,11,13
SLDT = Store local descriptor table register to register/memory	00001111 00000000 mod 000 r/m		2,3*	1	9

Shaded areas indicate instructions not available in 8086, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
PROTECTION CONTROL (Continued)					
LTR = Local task register from register/memory	00001111 00000000 mod 011 r/m		17,19*	1	9,11,13
STR = Store task register to register memory	00001111 00000000 mod 001 r/m		2,3*	1	9
LMSW = Load machine status word from register/memory	00001111 00000001 mod 110 r/m	3,6*	3,6*	2,3	9,13
SMSW = Store machine status word	00001111 00000001 mod 100 r/m	2,3*	2,3*	2,3	9
LAR = Load access rights from register/memory	00001111 00000010 mod reg r/m		14,16*	1	9,11,16
LSL = Load segment limit from register/memory	00001111 00000011 mod reg r/m		14,16*	1	9,11,16
ARPL = Adjust requested privilege level: from register/memory	01100011 mod reg r/m		10*,11*	2	8,9
VERR = Verify read access: register/memory	00001111 00000000 mod 100 r/m		14,16*	1	9,11,16
VERR = Verify write access:	00001111 00000000 mod 101 r/m		14,16*	1	9,11,16

Shaded areas indicate instructions not available in 8086, 88 microsystems.

Footnotes

The Effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

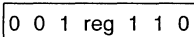
if mod = 11 then r/m is treated as a REG field
 if mod = 00 then DISP = 0*, disp-low and disp-high are absent
 if mod = 01 then DISP = disp-low sign-extended to 16 bits, disp-high is absent
 if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP
 if r/m = 001 then EA = (BX) + (DI) + DISP
 if r/m = 010 then EA = (BP) + (SI) + DISP
 if r/m = 011 then EA = (BP) + (DI) + DISP
 if r/m = 100 then EA = (SI) + DISP
 if r/m = 101 then EA = (DI) + DISP
 if r/m = 110 then EA = (BP) + DISP*
 if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EQ = disp-high: disp-low.

SEGMENT OVERRIDE PREFIX



reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DC

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

DATA SHEET REVISION REVIEW

The following list represents key differences between this and the -013 data sheet. Please review this summary carefully.

1. Package thermal specifications were added.
2. A graph of typical I_{CC} vs. Frequency for different output loads and Case Temperatures was added.
3. The Output Leakage Current Specification (I_{LO}) was changed.

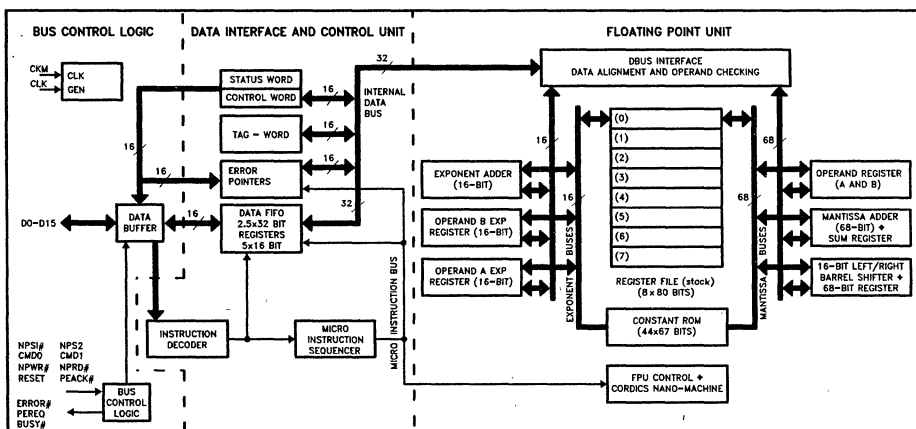
80C287A CHMOS III MATH COPROCESSOR

- High Performance 80-Bit Internal Architecture
 - Implements ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic
 - Implements Extended 80387 Instruction Set
 - Two to Three Times 8087/80287 Performance at Equivalent Clock Speed
 - Low Power Consumption
 - Upward Object-Code Compatible from 8087 and 80287
 - Interfaces with 80286 and 80C286 CPUs
 - Expands CPU's Data Types to Include 32-, 64-, 80-Bit Floating Point, 32-, 64-Bit Integers and 18-Digit BCD Operands
 - Directly Extends CPU's Instruction Set to Trigonometric, Logarithmic, Exponential, and Arithmetic Instructions for All Data Types
 - Full-Range 387™ DC Compatible Transcendental Operations for SINE, COSINE, TANGENT, ARCTANGENT, and LOGARITHM.
 - Built-In Exception Handling
 - Operates in Both Real and Protected Mode Systems
 - Eight 80-Bit Numeric Registers, Usable as Individually Addressable General Registers or as a Register Stack
 - Available in 40-pin DIP and 44-pin PLCC Package
- (See Packaging Outlines and Dimensions, order # 231369)

The Intel 80C287A CMOS Math CoProcessor is an extension to the Intel 80286 microprocessor architecture. It is functionally equivalent to the NMOS 80287, and provides higher speeds and low power consumption for battery powered or no-fan applications. When combined with an 80286 microprocessor the 80C287A dramatically increases the processing speed of computer application software which utilize mathematical operations. This makes an ideal computer workstation platform for applications such as financial modeling and spreadsheets, CAD/CAM, or graphics.

The 80C287A Math CoProcessor adds over seventy mnemonics to the 80286 microprocessor instruction set. Specific 80C287A math operations include logarithmic, arithmetic, exponential, and trigonometric functions. The 80C287A supports integer, extended integer, floating point and BCD data formats, and fully conforms to the ANSI/IEEE floating point standard.

The 80C287A is object code compatible with the 80287 and 8087. The 80C287A is fabricated with CHMOS III technology and available in a 40-pin DIP and 44-pin PLCC packages.



240347-1

Figure 0.1. 80C287A Block Diagram



80287 MATH COPROCESSOR

- High Performance 80-Bit Internal Architecture
- Implements Proposed IEEE Floating Point Standard 754
- Expands 80286 Data types to Include 32-, 64-, 80-Bit Floating Point, 32-, 64-Bit Integers and 18-Digit BCD Operands
- Object Code Compatible with 8087
- Built-in Exception Handling
- Operates in Both Real and Protected Mode 80286 Systems
- 8x80-Bit, Individually Addressable, Numeric Register Stack
- Protected Mode Operation Completely Conforms to the 80286 Memory Management and Protection Mechanisms
- Directly Extends 80286 Instruction Set to Trigonometric, Logarithmic, Exponential and Arithmetic Instructions for All Data types
- Operates with 80386 CPU without Software Modification
- Available in EXPRESS—Standard Temperature Range
- Available in 40 pin-CERDIP package
(see Packaging Spec: Order #231369)

The Intel 80287 Math CoProcessor is an extension to the Intel 80286 microprocessor architecture. When combined with the 80286 microprocessor the 80287 dramatically increases the processing speed of computer application software which utilize mathematical operations. This makes an ideal computer workstation platform for applications such as financial modeling and spreadsheets, CAD/CAM, or graphics.

The 80287 Math CoProcessor adds over seventy mnemonics to the 80286 microprocessor instruction set. Specific 80287 math operations include logarithmic arithmetic, exponential, and trigonometric functions. The 80287 supports integer, extended integer, floating point and BCD data formats, and fully conforms to the ANSI/IEEE floating point standard.

The 80286/80287 is object code compatible with the 8086/8087 and 8088/8087. The 80287 is fabricated with HMOS III technology and available in a 40-pin cerdip packages. A CMOS 80C287A math coprocessor is available for higher speed or low power applications.

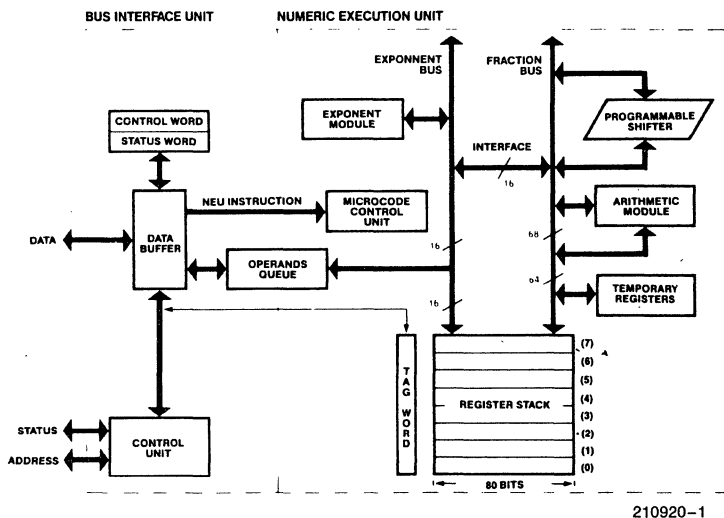
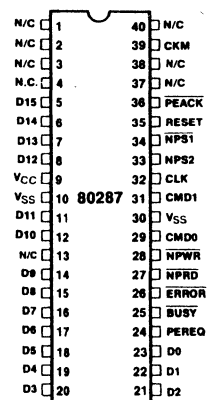


Figure 1. 80287 Block Diagram



210920-2

NOTE:
N/C Pins should not be connected

Figure 2.
80287 Pin Configuration

Table 1. 80287 Pin Description

Symbols	Type	Name and Function
CLK	I	CLOCK INPUT: this clock provides the basic timing for internal 80287 operations. Special MOS level inputs are required. The 82284 or 8284A CLK outputs are compatible to this input.
CKM	I	CLOCK MODE SIGNAL: indicates whether CLK input is to be divided by 3 or used directly. A HIGH input will cause CLK to be used directly. This input must be connected to V _{CC} or V _{SS} as appropriate. This input must be either HIGH or LOW 20 CLK cycles before RESET goes LOW.
RESET	I	SYSTEM RESET: causes the 80287 to immediately terminate its present activity and enter a dormant state. RESET is required to be HIGH for more than 4 80287 CLK cycles. For proper initialization the HIGH-LOW transition must occur no sooner than 50 μs after V _{CC} and CLK meet their D.C. and A.C. specifications.
D15–D0	I/O	DATA: 1-bit bidirectional data bus. Inputs to these pins may be applied asynchronous to the 80287 clock.
$\overline{\text{BUSY}}$	O	BUSY STATUS: asserted by the 80287 to indicate that it is currently executing a command.
$\overline{\text{ERROR}}$	O	ERROR STATUS: reflects the ES bit of the status word. This signal indicates that an unmasked error condition exists.
PEREQ	O	PROCESSOR EXTENSION DATA CHANNEL OPERAND TRANSFER REQUEST: a HIGH on this output indicates that the 80287 is ready to transfer data. PEREQ will be disabled upon assertion of $\overline{\text{PEACK}}$ or upon actual data transfer, whichever occurs first, if no more transfers are required.
$\overline{\text{PEACK}}$	I	PROCESSOR EXTENSION DATA CHANNEL OPERAND TRANSFER ACKNOWLEDGE: acknowledges that the request signal (PEREQ) has been recognized. Will cause the request (PEREQ) to be withdrawn in case there are no more transfers required. $\overline{\text{PEACK}}$ may be asynchronous to the 80287 clock.
NPRD	I	NUMERIC PROCESSOR READ: Enables transfer of data from the 80287. This input may be asynchronous to the 80287 clock.
NPWR	I	NUMERIC PROCESSOR READ: Enables transfer of data from the 80287. This input may be asynchronous to the 80287 clock.
NPS1, NPS2	I	NUMERIC PROCESSOR SELECTS: indicate the CPU is performing an ESCAPE instruction. Concurrent assertion of these signals (i.e., NPS1 is LOW and NPS2 is HIGH) enables the 80287 to perform floating point instructions. No data transfers involving the 80287 will occur unless the device is selected via these lines. These inputs may be asynchronous to the 80287 clock.
CMD1, CMD0	I	COMMAND LINES: These, along with select inputs, allow the CPU to direct the operation of the 80287. These inputs may be asynchronous to the 80287 clock.

Table 1. 80187 Pin Description (Continued)

Symbols	Type	Name and Function
V _{SS}	I	System ground, both pins must be connected to ground.
V _{CC}	I	+5V supply

FUNCTIONAL DESCRIPTION

The 80287 Numeric Processor Extension (NPX) provides arithmetic instructions for a variety of numeric data types in 80286/80287 systems. It also executes numerous built-in transcendental functions (e.g., tangent and log functions). The 80287 executes instructions in parallel with an 80286. It effectively

extends the register and instruction set of an 80286 system for existing 80286 data types and adds several new data types as well. Figure 3 presents the program visible register model of the 80286/80287. Essentially, the 80287 can be treated as an additional resource or an extension to the 80286 that can be used as a single unified system, the 80286/80287.

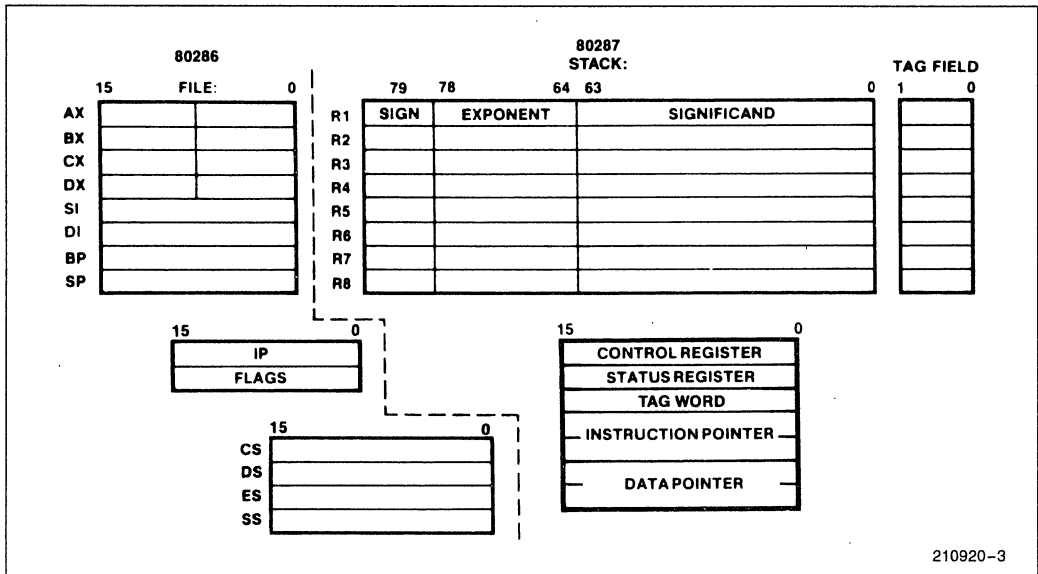


Figure 3. 80286/80287 Architecture

The 80287 has two operating modes similar to the two modes of the 80286. When reset, 80287 is in the real address mode. It can be placed in the protected virtual address mode by executing the SETPM ESC instruction. The 80287 cannot be switched back to the real address mode except by reset. In the real address mode, the 80286/80287 is completely software compatible with 8086/8087 and 8088/8087.

Once in protected mode, all references to memory for numerics data or status information, obey the 80286 memory management and protection rules giving a fully protected extension of the 80286 CPU. In the protected mode, 80286/80287 numerics software is also completely compatible with 8086/8087 and 8088/8087.

SYSTEM CONFIGURATION WITH 80286

As a processor extension to an 80286, the 80287 can be connected to the CPU as shown in Figure 4A. The data channel control signals (PEREQ, PEACK), the $\overline{\text{BUSY}}$ signal and the $\overline{\text{NPRD}}$, $\overline{\text{NPWR}}$ signals, allow the NPX to receive instructions and data from the CPU. When in the protected mode, all information received by the NPX is validated by the 80286 memory management and protection unit. Once started, the 80287 can process in parallel with and independent of the host CPU. When the NPX detects an error or exception, it will indicate this to the CPU by asserting the $\overline{\text{ERROR}}$ signal.

The NPX uses the processor extension request and acknowledge pins of the 80286 CPU to implement data transfers with memory under the protection model of the CPU. The full virtual and physical address space of the 80286 is available. Data for the 80287 in memory is addressed and represented in the same manner as for an 8087.

The 80287 can operate either directly from the CPU clock or with a dedicated clock. For operation with the CPU clock (CKM = 0), the 80287 works at one-third the frequency of the system clock (i.e., for an 8 MHz 80286, the 16 MHz system clock is divided down to 5.3 MHz). The 80287 provides a capability to internally divide the CPU clock by three to produce the required internal clock (33% duty cycle). To use a higher performance 80287 (8 MHz), an 8284A clock driver and appropriate crystal may be used to directly drive the 80287 with a $\frac{1}{3}$ duty cycle clock on the CLK input (CKM = 1). The following table describes the relationship between the clock speed and the 287 speed version needed as a function of the CKM state.

287 Speed Version	CLK Speed	
	CKM = 0	CKM = 1
5 MHz	12 MHz	5 MHz
6 MHz	16 MHz	6 MHz
8 MHz	20 MHz	8 MHz
10 MHz	25 MHz	10 MHz

SYSTEM CONFIGURATION WITH 80386

The 80287 can also be connected as a processor extension to the 80386 CPU as shown in Figure 4b. All software written for 8086/8087 and 80286/80287 is object code compatible with 80386/80287 and can benefit from the increased speed of the 80386 CPU.

Note that the $\overline{\text{PEACK}}$ input pin is pulled high. This is because the 80287 is not required to keep track of the number of words transferred during an operand transfer when it is connected to the 80386 CPU. Unlike the 80286 CPU, the 80386 CPU knows the exact length of the operand being transferred to/from the 80287. After an ESC instruction has been sent to the 80287, the 80386 processor extension data channel will initiate the data transfer as soon as it receives the PEREQ signal from the 80287. The transfer is automatically terminated by the 80386 CPU as soon as all the words of the operand have been transferred.

Because of the very high speed local bus of the 80386 CPU, the 80287 cannot reside directly on the CPU local bus. A local bus controller logic is used to generate the necessary read and write cycle timings as well as the chip select timings for the 80287. The 80386 CPU uses I/O addresses 800000F8 through 800000FF to communicate with the 80287. This is beyond the normal I/O address space of the CPU and makes it easier to generate the chip select signals using A31 and $\overline{\text{M}/\overline{\text{IO}}}$. It may also be noted that the 80386 CPU automatically generates 16-bit bus cycles whenever it communicates with the 80287.

HARDWARE INTERFACE

Communication of instructions and data operands between the 80286 and 80287 is handled by the $\overline{\text{CMD0}}$, $\overline{\text{CMD1}}$, $\overline{\text{NPS1}}$, $\overline{\text{NPS2}}$, $\overline{\text{NPRD}}$, and $\overline{\text{NPWR}}$ signals. I/O port addresses 00F8H, 00FAH, and 00FCH are used by the 80286 for this communication. When any of these addresses are used, the $\overline{\text{NPS1}}$ input must be LOW and $\overline{\text{NPS2}}$ input HIGH. The $\overline{\text{IORC}}$ and $\overline{\text{IOWC}}$ outputs of the 82288 identify I/O space transfers (see Figure 4A). $\overline{\text{CMD0}}$ should be connected to latched 80286 A1 and $\overline{\text{CMD1}}$ should be connected to latched 80286 A2.

I/O ports 00F8H to 00FFH are reserved for the 80286/80287 interface. To guarantee correct operation of the 80287, programs must not perform any I/O operations to these ports.

The PEREQ, $\overline{\text{PEACK}}$, $\overline{\text{BUSY}}$, and $\overline{\text{ERROR}}$ signals of the 80287 are connected to the same-named 80286 input. The data pins of the 80287 should be directly connected to the 80286 data bus. Note that all bus drivers connected to the 80286 local bus must be inhibited when the 80286 reads from the 80287. The use of $\overline{\text{M}/\overline{\text{IO}}}$ in the decoder prevents INTA bus cycles from disabling the data transceivers.

PROGRAMMING INTERFACE

Table 2 lists the seven data types the 80287 supports and presents the format for each type. These

values are stored in memory with the least significant digits at the lowest memory address. Programs retrieve these values by generating the lowest address. All values should start at even addresses for maximum system performance.

Internally the 80287 holds all numbers in the temporary real format. Load instructions automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating point number or

18-digit packed BCD numbers into temporary real format. Store instructions perform the reverse type conversion.

80287 computations use the processor's register stack. These eight 80-bit registers provide the equivalent capacity of 40 16-bit registers. The 80287 register set can be accessed as a stack, with instructions operating on the top one or two stack elements, or as a fixed clock set, with instructions operating on explicitly designated registers.

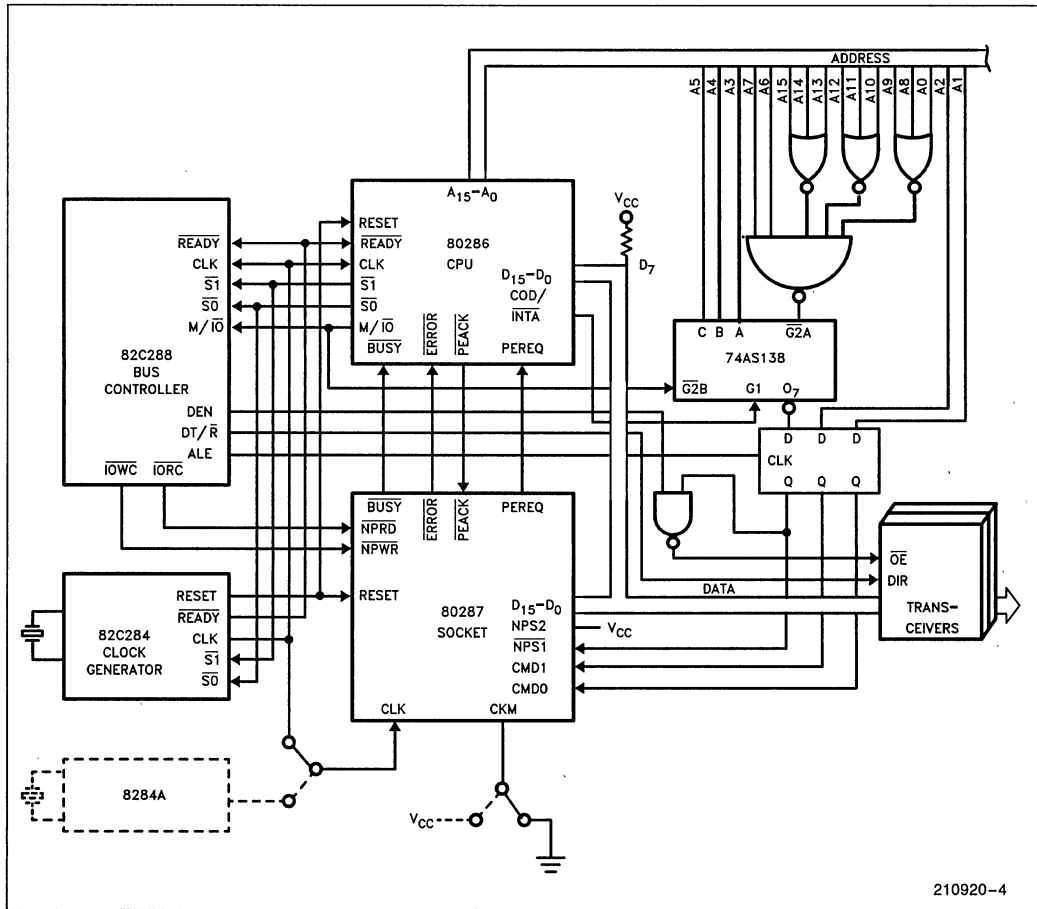
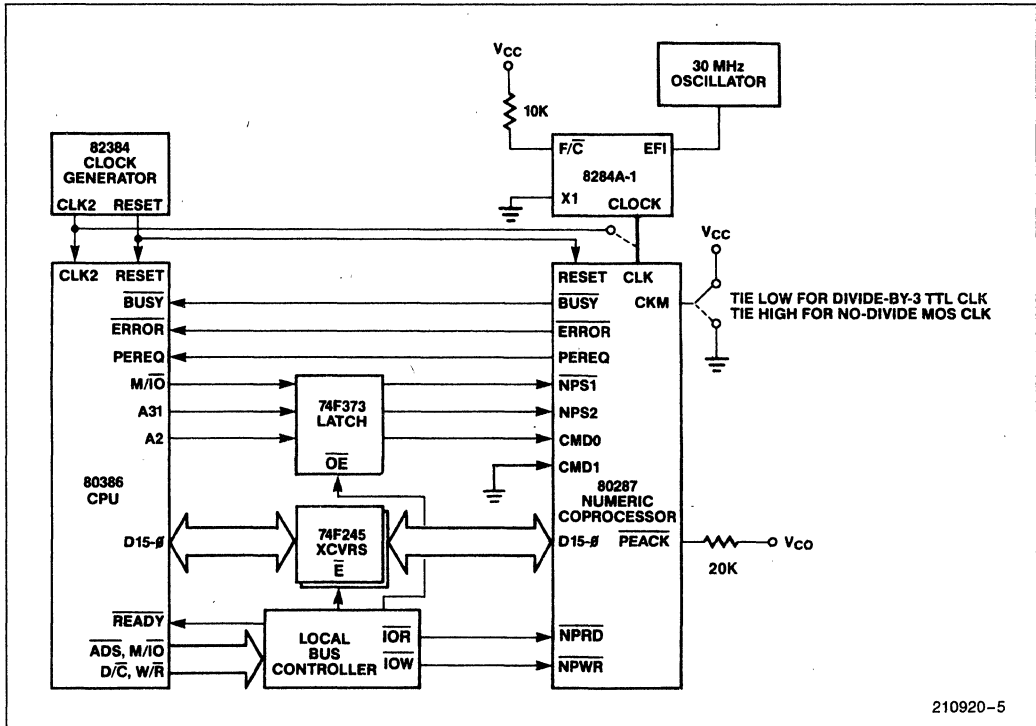


Figure 4A. 80286/80287 System Configuration



210920-5

Figure 4B. 80386/80287 System Configuration

Table 2. 80287 Data Type Representation in Memory

Data Formats	Range	Precision	Most Significant Byte				HIGHEST ADDRESSED BYTE							
			7	0	7	0	7	0	7	0	7	0	7	0
Word Integer	10^4	16 Bits												
Short Integer	10^9	32 Bits												
Long Integer	10^{19}	64 Bits												
Packed BCD	10^{18}	18 Digits												
Short Real	$10^{\pm 38}$	24 Bits												
Long Real	$10^{\pm 308}$	53 Bits												
Temporary Real	$10^{\pm 4932}$	64 Bits												

210920-6

NOTES:

1. S = Sign bit (0 = positive, 1 = negative)
2. d_n = Decimal digit (two per byte)
3. X = Bits have no significance; 8087 ignores when loading, zeros when storing.
4. ▲ = Position of implicit binary point
5. I = Integer bit of significant; stored in temporary real, implicit in short and long real.
6. Exponent Bias (normalized values):
 Short Real: 127 (7FH)
 Long Real: 1023 (3FFH)
 Temporary Real: 16383 (3FFFH)
7. Packed BCD: $(-1)^s (D_{17} \dots D_0)$
8. Real: $(-1)^s (2^E \text{-BIAS})(F_0 F_1 \dots)$

Table 6 lists the 80287's instructions by class. No special programming tools are necessary to use the 80287 since all new instructions and data types are directly supported by the 80286 assembler and

appropriate high level languages. All 8086/8088 development tools which support the 8087 can also be used to develop software for the 80286/80287 in real address mode.

SOFTWARE INTERFACE

The 80286/80287 is programmed as a single processor. All communication between the 80286 and the 80287 is transparent to software. The CPU automatically controls the 80287 whenever a numeric instruction is executed. All memory addressing modes, physical memory, and virtual memory of the CPU are available for use by the NPX.

Since the NPX operates in parallel with the CPU, any errors detected by the NPX may be reported after the CPU has executed the ESCAPE instruction which caused it. To allow identification of the failing numeric instruction, the NPX contains two pointer registers which identify the address of the failing numeric instruction and the numeric memory operand if appropriate for the instruction encountering this error.

INTERRUPT DESCRIPTION

Several interrupts of the 80286 are used to report exceptional conditions while executing numeric programs in either real or protected mode. The interrupts and their functions are shown in Table 3.

PROCESSOR ARCHITECTURE

As shown in Figure 1, the NPX is internally divided into two processing elements, the bus interface unit (BIU) and the numeric execution unit (NEU). The NEU executes all numeric instructions, while the BIU receives and decodes instructions, requests operand transfers to and from memory and executes processor control instructions. The two units are able to operate independently of one another allowing the BIU to maintain asynchronous communication with the CPU while the NEU is busy processing a numeric instruction.

BUS INTERFACE UNIT

The BIU decodes the ESC instruction executed by the CPU. If the ESC code defines a math instruction, the BIU transmits the formatted instruction to the NEU. If the ESC code defines an administrative instruction, the BIU executes it independently of the NEU. The parallel operation of the NPX with the CPU is normally transparent to the user. The BIU generates the `BUSY` and `ERROR` signals for 80286/80287 processor synchronization and error notification, respectively.

The 80287 executes a single numeric instruction at a time. When executing most ESC instructions, the

Table 3. 80286 Interrupt Vectors Reserved for NPX

Interrupt Number	Interrupt Function
7	An ESC instruction was encountered when EM or TS of the 80286 MSW was set. EM = 1 indicates that software emulation of the instruction is required. When TS is set, either an ESC or WAIT instruction will cause interrupt 7. This indicates that the current NPX context may not belong to the current task.
9	The second or subsequent words of a numeric operand in memory exceeded a segment's limit. This interrupt occurs after executing an ESC instruction. The saved return address will not point at the numeric instruction causing this interrupt. After processing the addressing error, the 80286 program can be restarted at the return address with IRET. The address of the failing numeric instruction and numeric operand and saved in the 80287. An interrupt handler for this interrupt <i>must</i> execute FNINIT before <i>any</i> other ESC or WAIT instruction.
13	The starting address of a numeric operand is not in the segment's limit. The return address will point at the ESC instruction, including prefixes, causing this error. The 80287 has not executed this instruction. The instruction and data address is 80287 refer to a previous, correctly executed, instruction.
16	The previous numeric instruction caused an unmasked numeric error. The address of the faulty numeric instruction or numeric data operand is stored in the 80287. Only ESC or WAIT instructions can cause this interrupt. The 80286 return address will point at a WAIT or ESC instruction, including prefixes, which may be restarted after clearing the error condition in the NPX.

80286 tests the $\overline{\text{BUSY}}$ pin and waits until the 80287 indicates that it is not busy before initiating the command. Once initiated, the 80286 continues program execution while the 80287 executes the ESC instruction. In 8086/8087 systems, this synchronization is achieved by placing a WAIT instruction before an ESC instruction. For most ESC instructions, the 80287 does not require a WAIT instruction before the ESC opcode. However, the 80287 will operate correctly with these WAIT instruction. In all cases, a WAIT or ESC instruction should be inserted after any 80287 store to memory (except FSTSW and FSTCW) or load from memory (except FLDENV or FRSTOR) before the 80286 reads or changes the value to be sure the numeric value has already been written or read by the NPX.

Data transfers between memory and the 80287, when needed, are controlled by the PEREQ PEACK, $\overline{\text{NPRD}}$, $\overline{\text{NPWR}}$, $\overline{\text{NPST1}}$, $\overline{\text{NPS2}}$ signals. The 80286 does the actual data transfer with memory through its processor extension data channel. Numeric data transfers with memory performed by the 80286 use the same timing as any other bus cycle. Control signals for the 80287 are generated by the 80826 as

shown in Figure 4a, and meet the timing requirements shown in the AC requirements section.

NUMERIC EXECUTION UNIT

The NEU executes all instructions that involve the register stack; these include arithmetic, logical, transcendental, constant and data transfer instructions. The data path in the NEU is 84 bits wide (68 significant bits, 15 exponent bits and a sign bit) which allows internal operand transfers to be performed at very high speeds.

When the NEU begins executing an instruction, it activated the BIU BUSY signal. This signal is used in conjunction with the CPU WAIT instruction or automatically with most of the ESC instructions to synchronize both processors.

REGISTER SET

The 80287 register set is shown in Figure 5. Each of the eight data registers in the 8087's register stack

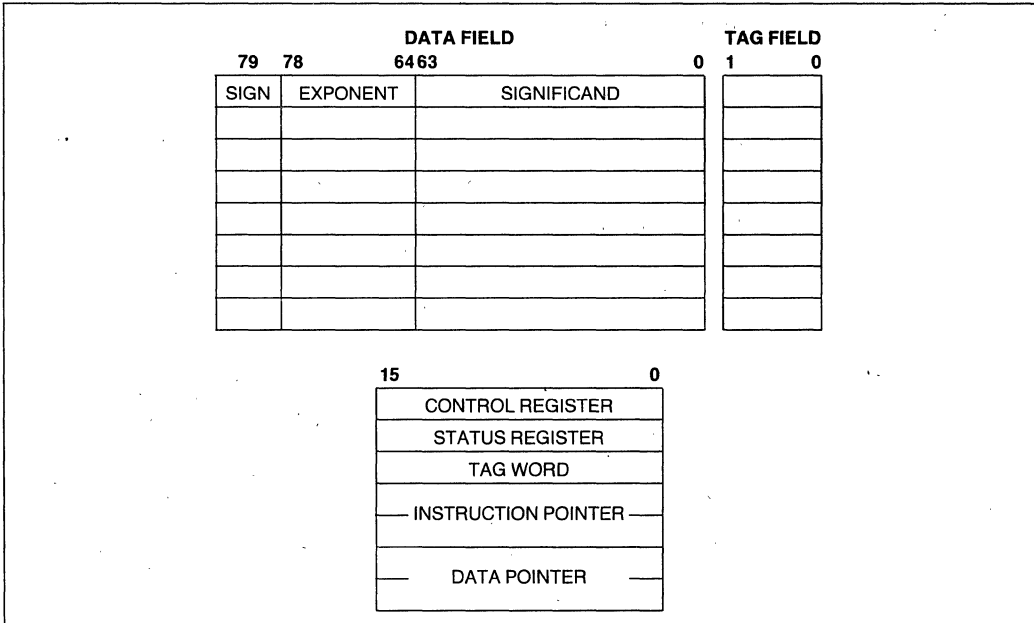


Figure 5. 80287 Register Set

is 80 bits wide and is divided into "fields" corresponding to the NPX's temporary real data type.

At a given point in time the TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by 1 and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by 1. Like 80286 stacks in memory, the 80287 register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register pointed by the TOP. Other instructions allow the programmer to explicitly specify the register which is to be used. This explicit register addressing is also "top-relative."

STATUS WORD

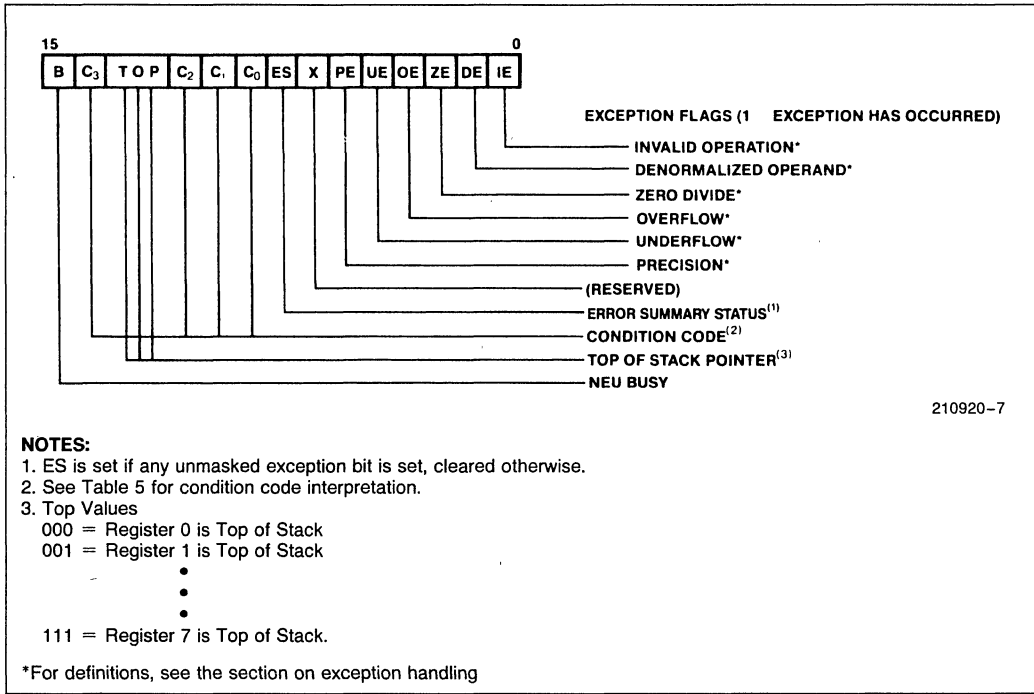
The 16-bit status word (in the status register) shown in Figure 6 reflects the overall state of the 80287. It may be read and inspected by CPU code. The busy bit (bit 15) indicates whether the NEU is executing an instruction (B = 1) or is idle (B = 0).

The instructions FSTSW, FSTSW AX, FSTENV, and FSAVE which store the status word are executed exclusively by the BIU and do not set the busy bit themselves or require the Busy bit be cleared in order to be executed.

The four numeric condition code bits (C₀-C₃) are similar to the flags in a CPU: instructions that perform arithmetic operations update these bits to reflect the outcome of NPX operations. The effect of these instructions on the condition code is summarized in Tables 4a and 4b.

Bits 14-12 of the status word point to the 80287 register that is the current top-of-stack (TOP) as described above. Figure 6 shows the six error flags in bits 5-0 of the status word. Bits 5-0 are set to indicate that the NEU has detected an exception while executing an instruction. The section on exception handling explains how they are set and used.

Bit 7 is the error summary status bit. This bit is set if any unmasked exception bit is set and cleared otherwise. If this bit is set, the ERROR signal is asserted.



210920-7

NOTES:

- 1. ES is set if any unmasked exception bit is set, cleared otherwise.
- 2. See Table 5 for condition code interpretation.
- 3. Top Values
 - 000 = Register 0 is Top of Stack
 - 001 = Register 1 is Top of Stack
 -
 -
 -
 - 111 = Register 7 is Top of Stack.

*For definitions, see the section on exception handling

Figure 6. 80287 Status Word

TAG WORD

The tag word marks the content of each register as shown in Figure 7. The principal function of the tag word is to optimize the NPX's performance. The eight two-bit tags in the tag word can be used, however, to interpret the contents of 80287 registers.

INSTRUCTION AND DATA POINTERS

The instruction and data pointers (See Figures 8a and 8b) are provided for user-written error handlers. Whenever the 80287 executes a new instruction, the BIU saves the instruction address, the operand address (if present) and the instruction opcode. 80287 instructions can store this data into memory.

The instruction and data pointers appear in one of two formats depending on the operating mode of the 80287. In real mode, these values are the 20-bit physical address and 11-bit opcode formatted like the 8087. In protection mode, these values are the

32-bit virtual address used by the program which executed an ESC instruction. The same FLDENV/ FSTENV/ FSAVE/ FRSTOR instructions as those of the 8087 are used to transfer these values between the 80287 registers and memory.

The saved instruction address in the 80287 will point at any prefixes which preceded the instruction. This is different than in the 8087 which only pointed at the ESCAPE instruction opcode.

CONTROL WORD

The NPX provides several processing options which are selected by loading a word from memory into the control word. Figure 9 shows the format and encoding of fields in the control word.

The low order byte of this control word configures the 80287 error and exception masking. Bits 5-0 of the control word contain individual masks for each of the six exceptions that the 80287 recognizes. The high order byte of the control word configures the

Table 4a. Condition Code Interpretation

Instruction Type	C ₃	C ₂	C ₁	C ₀	Interpretation
Compare, Test	0	0	X	0	ST > Source or 0 (FTST)
	0	0	X	1	ST < Source or 0 (FTST)
	1	0	X	0	ST = Source or 0 (FTST)
	1	1	X	1	ST is not comparable
Remainder	Q ₁	0	Q ₀	Q ₂	Complete reduction with three low bits of quotient (See Table 5b)
	U	1	U	U	Incomplete Reduction
Examine	0	0	0	0	Valid, positive unnormalized
	0	0	0	1	Invalid, positive, exponent = 0
	0	0	1	0	Valid, negative, unnormalized
	0	0	1	1	Invalid, negative, exponent = 0
	0	1	0	0	Valid, positive, normalized
	0	1	0	1	Infinity, positive
	0	1	1	0	Valid, negative, normalized
	0	1	1	1	Infinity, negative
	1	0	0	0	Zero, positive
	1	0	0	1	Empty
	1	0	1	0	Zero, Negative
	1	0	1	1	Empty
	1	1	0	0	Invalid, positive, exponent = 0
	1	1	0	1	Empty
1	1	1	0	Invalid, negative, exponent = 0	
1	1	1	1	Empty	

NOTES:

1. ST = Top of Stack
2. X = value is not affected by instruction
3. U = value is undefined following instruction
4. Q_n = Quotient bit n

Table 4b. Condition Code Interpretation after FPREM (See Note 1) Instruction as a Function of Dividend Value

Dividend Range	Q ₂	Q ₁	Q ₀
Dividend < 2 * Modulus	C ₃	C ₁	Q ₀
Dividend < 4 * Modulus	C ₃	Q ₁	Q ₀
Dividend ≥ 4 * Modulus	Q ₂	Q ₁	Q ₀

NOTE:

1. Previous value of indicated bit, not affected by FPREM instruction execution.

80287 operating mode including precision, rounding, and infinity control. The precision control bits (bits 9–8) can be used to set the 80287 internal operating precision at less than the default of temporary real (80-bit) precision. This can be useful in providing compatibility with the early generation arithmetic processors of smaller precision than the 80287. The rounding control bits (bits 11–10) provide for directed rounding and true chop as well as the unbiased round to nearest even mode specified in the IEEE standard. Control over closure of the number space at infinity is also provided (either affine closure: $\pm \infty$, or projective closure: ∞ , is treated as unsigned, may be specified).

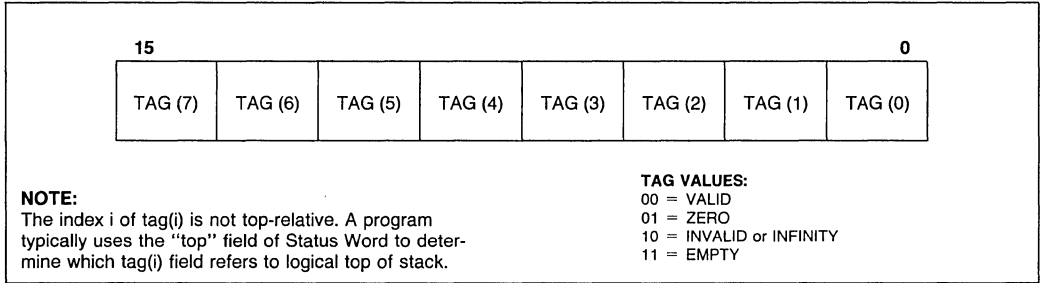


Figure 7. 80287 Tag Word

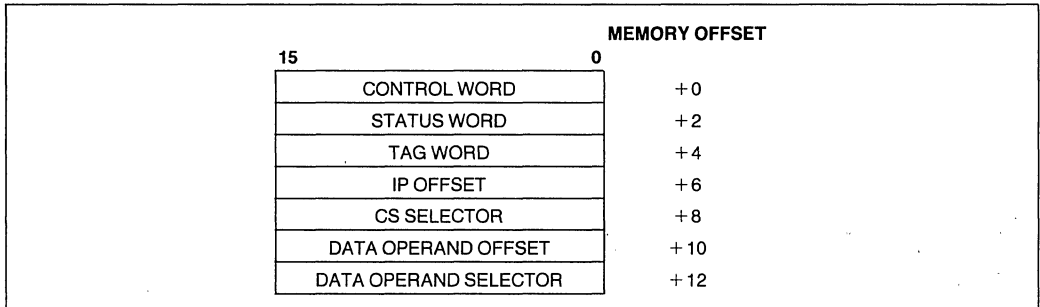


Figure 8a. Protected Mode 80287 Instruction and Data Pointer Image in Memory

EXCEPTION HANDLING

The 80287 detects six different exception conditions that can occur during instruction execution. Any or all exceptions will cause the assertion of external ERROR signal and ES bit of the Status Word if the appropriate exception masks are not set.

The exceptions that the 80287 detects and the 'default' procedures that will be carried out if the exception is masked, are as follows:

Invalid Operation: Stack overflow, stack underflow, indeterminate form (0/0, ∞, -∞, etc) or the use of a Non-Number (NaN) as an operand. An exponent value of all ones and non-zero significand is reserved to identify NaNs. If this exception is masked, the 80287 default response is to generate a specific

NAN called INDEFINITE, or to propagate already existing NaNs as the calculation result.

Overflow: The result is too large in magnitude to fit the specified format. The 80287 will generate an encoding for infinity if this exception is masked.

Zero Divisor: The divisor is zero while the dividend is a non-infinite, non-zero number. Again, the 80287 will generate an encoding for infinity if this exception is masked.

Underflow: The result is non-zero but too small in magnitude to fit in the specified format. If this exception is masked the 80287 will denormalize (shift right) the fraction until the exponent is in range. The process is called gradual underflow.

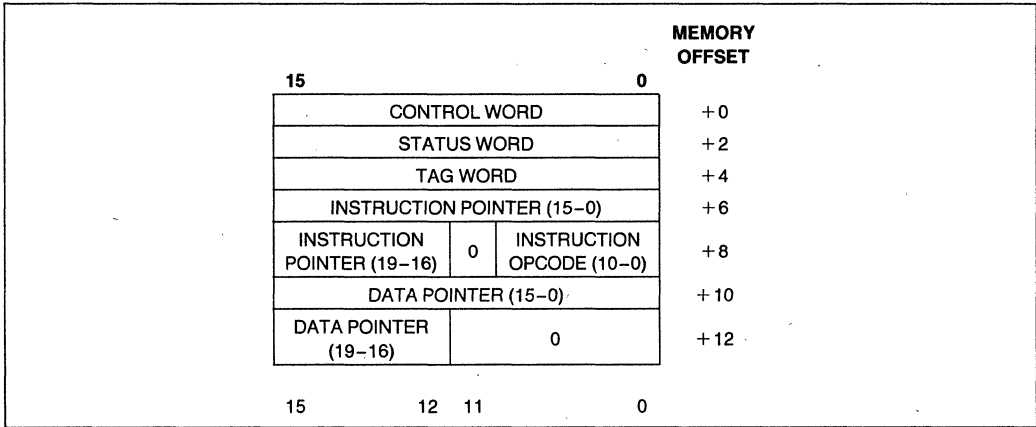


Figure 8b. Real Mode 80287 Instruction and Data Pointer Image in Memory

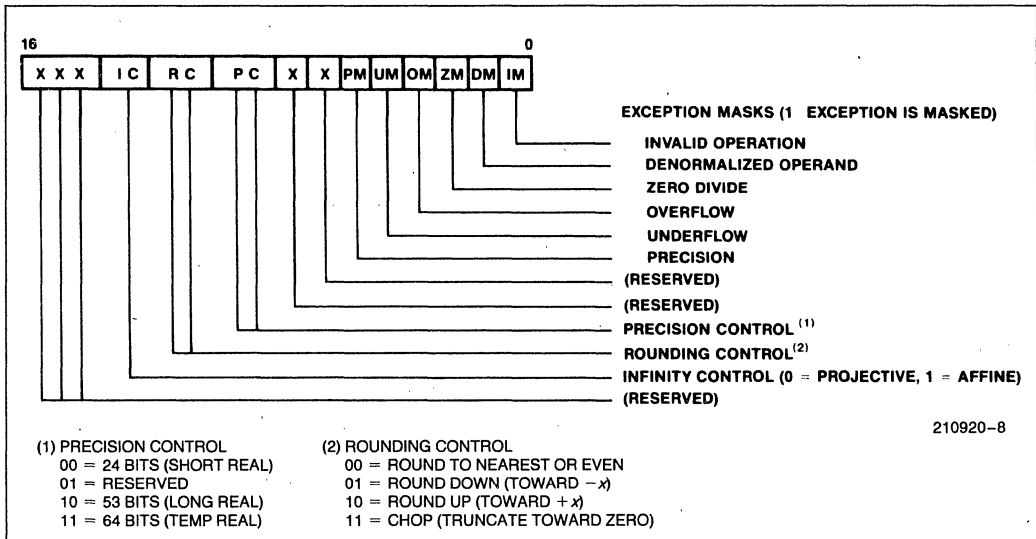


Figure 9. 80287 Control Word

Denormalized Operand: At least one of the operands is denormalized; it has the smallest exponent but a non-zero significand. Normal processing continues if this exception is masked off.

Inexact Result: The true result is not exactly representable in the specified format, the result is rounded according to the rounding mode, and this flag is set. If this exception is masked, processing will simply continue.

If the error is not masked, the corresponding error bit and the error status bit (ES) in the control word will be set, and the $\overline{\text{ERROR}}$ output signal will be asserted. If the CPU attempts to execute another ESC or WAIT instruction, exception 7 will occur.

The error condition must be resolved via an interrupt service routine. The 80287 saves the address of the floating point instruction causing the error as well as the address of the lowest memory location of any memory operand required by that instruction.

8086/8087 COMPATIBILITY:

The 80286/80287 supports portability of 8086/8087 programs when it is in the real address mode. However, because of differences in the numeric error handling techniques, error handling routines *may* need to be changed. The differences between an 80286/80287 and 8086/8087 are:

1. The NPX error signal does not pass through an interrupt controller (8087 INT signal does).

Therefore, any interrupt controller oriented instructions for the 8086/8087 may have to be deleted.

2. Interrupt vector 16 must point at the numeric error handler routine.
3. The saved floating point instruction address in the 80287 includes any leading prefixes before the ESCAPE opcode. The corresponding saved address of the 8087 does not include leading prefixes.
4. In protected mode, the format of the saved instruction and operand pointers is different than for the 8087. The instruction opcode is not saved—it must be read from memory if needed.
5. Interrupt 7 will occur when executing ESC instructions with either TS or EM or MSW = 1. If TS or MSW = 1 then WAIT will also cause interrupt 7. An interrupt handler should be added to handle this situation.
6. Interrupt 9 will occur if the second or subsequent words of a floating point operand fall outside a segment's size. Interrupt 13 will occur if the starting address of a numeric operand falls outside a segment's size. An interrupt handler should be added to report these programming errors.

In the protected mode, 8086/8087 application code can be directly ported via recompilation if the 80286 memory protection rules are not violated.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias0°C to 70°C
 Storage Temperature -65°C to +150°C
 Case Temperature0°C to 85°C
 Voltage on any Pin with
 Respect to Ground -1.0 to +7V
 Power Dissipation3.0 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $T_A = 0^\circ\text{C}$ to 70°C , $T_C = 0^\circ\text{C}$ to 85°C , $V_{CC} = 5\text{V} \pm 5\%$
ALL SPEEDS SELECTIONS

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL}	Input LOW Voltage	-0.5	0.8	V	
V_{IH}	Input HIGH Voltage	2.0	$V_{CC} + 0.5$	V	
V_{IHC}	Clock Input HIGH Voltage CKM = 1: CKM = 0:	2.0	$V_{CC} + 1$	V	
		3.8	$V_{CC} + 1$	V	
V_{ILC}	Clock Input LOW Voltage CKM = 1 CKM = 0	-0.5	0.8	V	
		-0.5	0.6	V	
V_{OL}	Output LOW Voltage		0.45	V	$I_{OL} = 3.0\text{ mA}$
V_{OH}	Output HIGH Voltage	2.4		V	$I_{OH} = -400\ \mu\text{A}$
I_{LI}	Input Leakage Current	•	± 10	μA	$0\text{V} \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current	•	± 10	μA	$0.45\text{V} \leq V_{OUT} \leq V_{CC}$
I_{CC}	Power Supply Current	•	600	mA	$T_A = 0^\circ\text{C}$
			475	mA	$T_A = 25^\circ\text{C}$
			375	mA	$T_A = 70^\circ\text{C}$
C_{IN}	Input Capacitance	•	10	pF	$F_C = \text{MHz}$
C_O	Input/Output Capacitance (D0-D15)	•	20	pF	$V_C = 1\text{ MHz}$
C_{CLK}	CLK Capacitance	•	12	pF	$F_C = 1\text{ MHz}$

A.C. CHARACTERISTICS $T_A = 0^{\circ}\text{C}$ to 70°C , $T_{\text{CASE}} = 0^{\circ}\text{C}$ to 85°C , $V_{\text{CC}} = 5\text{V} \pm 5\%$
TIMING REQUIREMENTS

A.C. timings are referenced to 0.8V and 2.0V points on signals unless otherwise noted.

Symbol	Parameter	80287-6 6 MHz		80287-8 8 MHz		80287-10 10 MHz		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
T _{CLCL}	CLK Period CKM = 1: CKM = 0:	166	500	125	500	100	500	ns ns	
		62.5	166	50	166	40	166		
T _{CLCH}	CLK LOW Time CKM = 1: CKM = 0:	100	343	68	343	62	343	ns ns	At 0.8V At 0.6V
		15	146	15	146	11	146		
T _{CHCL}	CLK HIGH Time CKM = 1: CKM = 0:	50	230	43	230	28	230	ns ns	At 2.0V At 3.6V
		20	151	20	151	18	151		
T _{CH1CH2}	CLK Rise Time		10		10		10	ns	1.0V to 3.6V if CKM = 0
T _{CL2CL1}	CLK Fall Time		10		10		10	ns	3.6V to 1.0V if CKM = 0
T _{DYWH}	Data Setup to NPWR Inactive	75		75		75		ns	
T _{WHDX}	Data Hold from NPWR Inactive	30		18		18		ns	
T _{WLWH} T _{RLRH}	NPWR NPRD Active Time	95		90		90		ns	At 0.8V
T _{AVWL} T _{AVRL}	Command Valid to NPWR or NPRD Active	0		0		0		ns	
T _{MHRL}	Minimum Delay from PEREQ Active to NPRD Active	130		130		100		ns	
T _{KLKH}	PEAK Active Time	85		85		60		ns	At 0.8V
T _{KHKL}	PEAK Inactive Time	250		250		200		ns	At 2.0V
T _{KHCH}	PEAK Inactive to NPWR, NPRD Inactive	50		40		40		ns	
T _{CHKL}	NPWR, NPRD Inactive to PEAK Active	-30		-30		-30		ns	
T _{WHAX} T _{RHAX}	Command Hold from NPWR, NPRD Inactive	30		30		22		ns	
T _{KLCL}	PEAK Active Setup to NPWR NPRD Active	50		40		40		ns	

A.C. CHARACTERISTICS $T_A = 0^\circ\text{C to }70^\circ\text{C}$, $T_{\text{CASE}} = 0^\circ\text{C to }85^\circ\text{C}$, $V_{\text{CC}} = 5\text{V} \pm 5\%$ (Continued)

TIMING REQUIREMENTS (Continued)

A.C. timings are referenced to 0.8V and 2.0V points on signals unless otherwise noted.

Symbol	Parameter	80287-6 6 MHz		80287-8 8 MHz		80287-10 10 MHz		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
T_{IVCL}	NPWR, NPRD to CLK Setup Time	70		70		53		ns	(Note 1)
T_{CLIH}	NPWR, NPRD from CLK Hold Time	45		45		37		ns	(Note 1)
T_{RSCL}	RESET to CLK Setup Time	20		20		20		ns	(Note 1)
T_{CLRS}	RESET from CLK Hold Time	20		20		20		ns	(Note 1)

TIMING RESPONSES

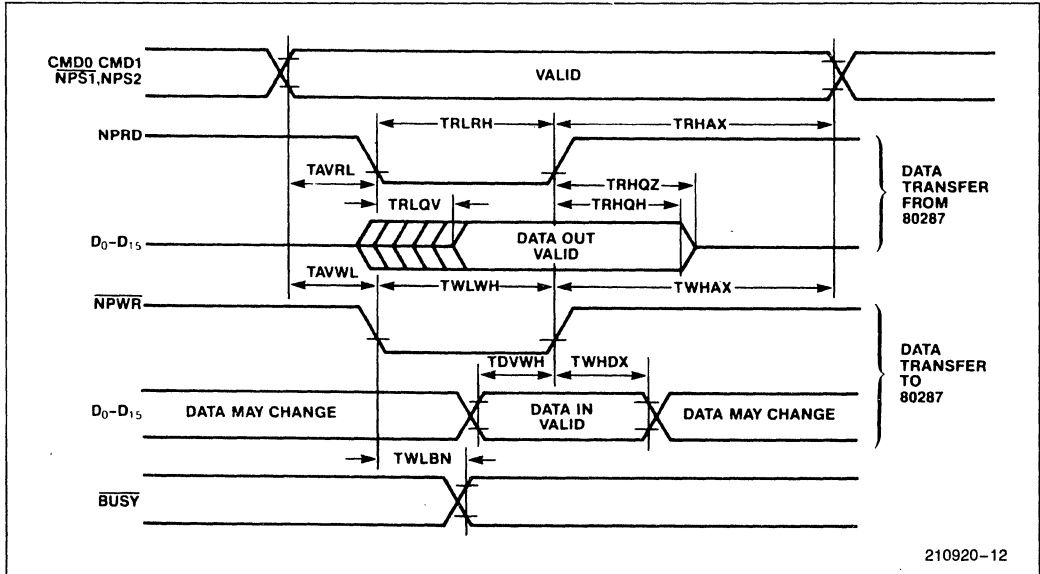
Symbol	Parameter	80287-6 6 MHz		80287-8 8 MHz		80287-10 10 MHz		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
T_{RHQZ}	NPRD Inactive to Data Float		37.5		35		21	ns	(Note 2)
T_{RLOV}	NPRD Active to Data Valid		60		60		60	ns	(Note 3)
T_{ILBH}	ERROR Active to BUSY Inactive	100		100		100		ns	(Note 4)
T_{WLBV}	NPWR Active to BUSY Active		100		100		100	ns	(Note 5)
T_{KLML}	PEAK Active to PEREQ Inactive		127		127		100	ns	(Note 6)
T_{CMDI}	Command Inactive Time								
	Write-to-Write	95		95		75		ns	At 2.0V
	Read-to-Read	95		95		75		ns	At 2.0V
	Write-to-Read	95		95		75		ns	At 2.0V
	Read-to-Write	95		95		75		ns	At 2.0V
T_{RHQH}	Data Hold from NPRD Inactive	3		3		3		ns	(Note 7)

NOTES:

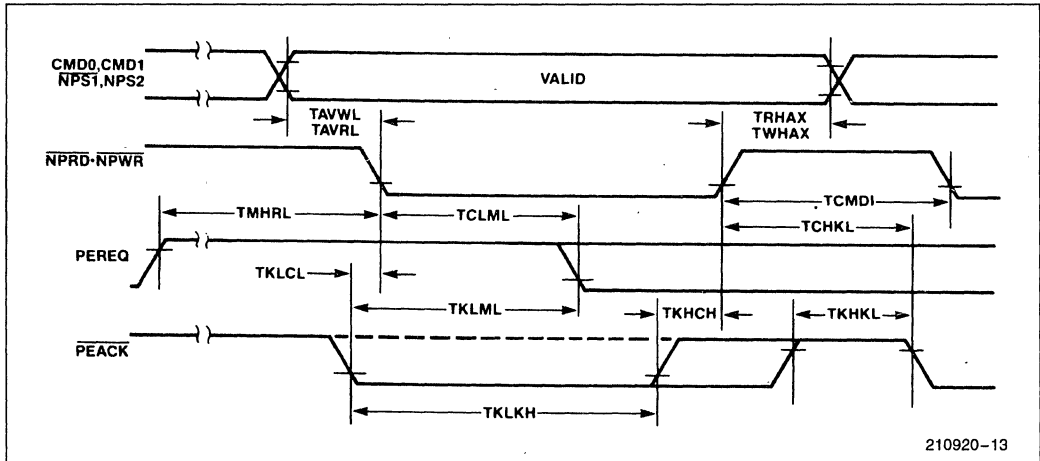
- This is an asynchronous input. This specification is given for testing purposes only, to assure recognition at a specific CLK edge.
- Float condition occurs when output current is less than I_{LO} on D0–D15.
- D0–D15 I_{OSINF} : $X_L = 100\text{ pF}$.
- BUSY loading: $CL = 100\text{ pF}$.
- BUSY loading: $CL = 100\text{ pF}$.
- On last data transfer on numeric instruction.
- D0–D15 loading: $CL = 100\text{ pF}$.

WAVEFORMS

DATA TRANSFER TIMING (Initiated by 80286)

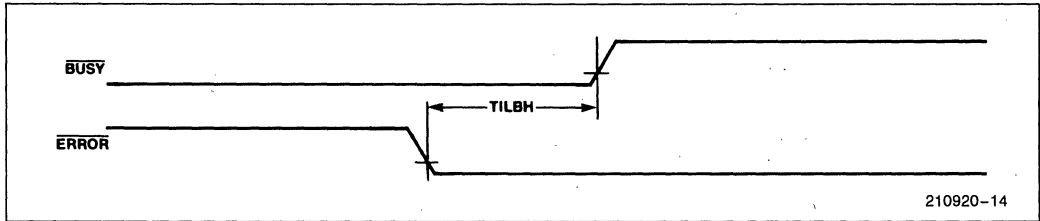


DATA CHANNEL TIMING (Initiated by 80287)

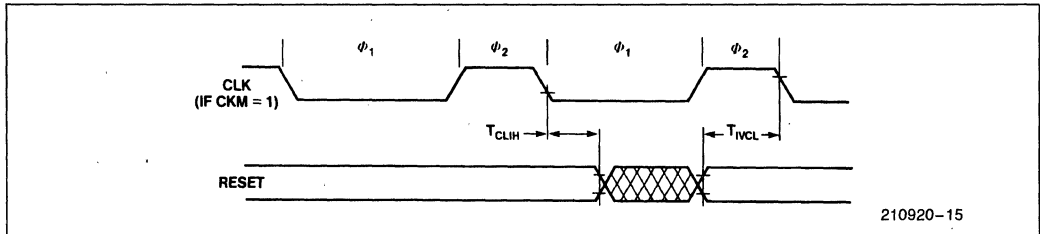


WAVEFORMS (Continued)

ERROR OUTPUT TIMING



CLK, RESET TIMING (CKM = 1)

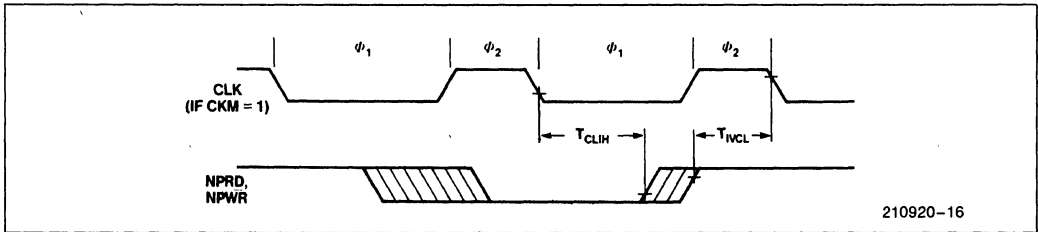


NOTE:

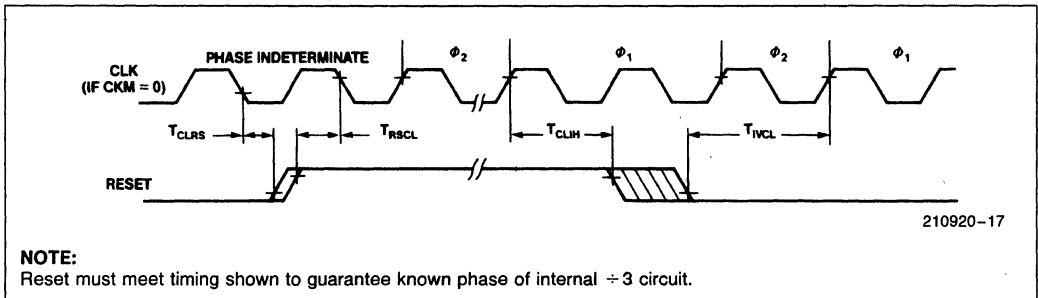
Reset, \overline{NPWR} , \overline{NPRD} are inputs asynchronous to CLK. Timing requirements on this page are given for testing purposes only, to assure recognition at a specific CLK edge.

WAVEFORMS (Continued)

CLK, $\overline{\text{NPRD}}$, $\overline{\text{NPWR}}$ TIMING (CKM = 1)



CLK, RESET TIMING (CKM = 0)



CLK, $\overline{\text{NPRD}}$, $\overline{\text{NPWR}}$ TIMING (CKM = 0)

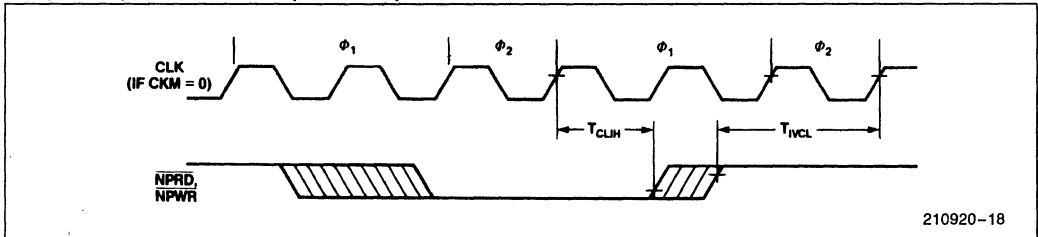
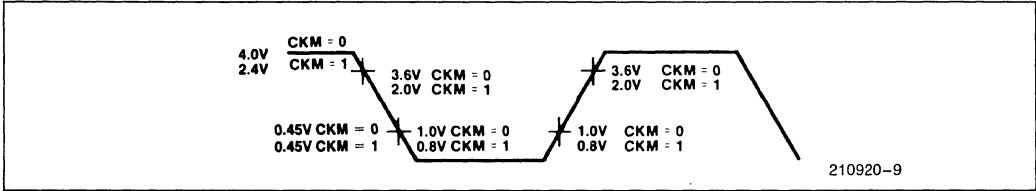
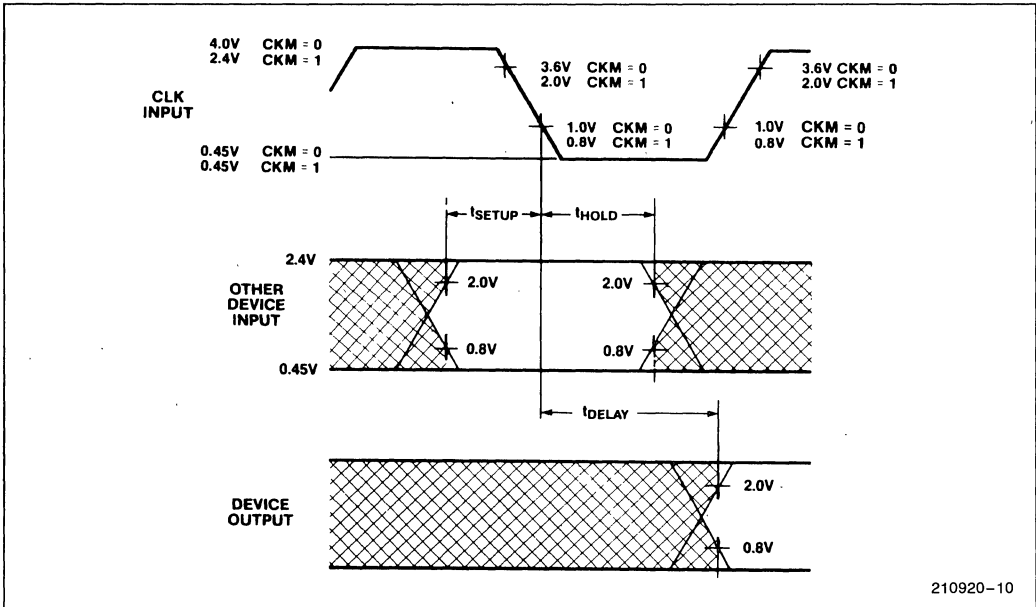


Table 6. 80287 Extensions to the 80286 Instruction Set

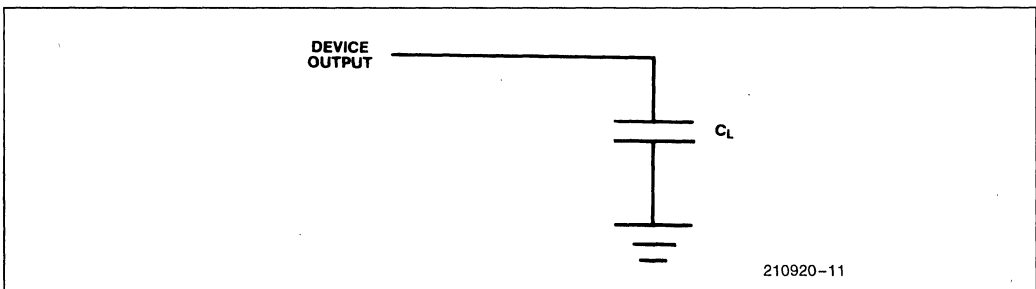
Data Transfer	Optional 8,16 Bit Displacement	Clock Count Range				
		32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer	
FLD = LOAD	MF =	00	01	10	11	
Integer/Real Memory to ST(0)	ESCAPE MF 1 MOD 0 0 0 R/M	DISP	38-56	52-60	40-60	46-54
Long Integer Memory to ST(0)	ESCAPE 1 1 1 MOD 1 0 1 R/M	DISP	60-68			
Temporary Real Memory to ST(0)	ESCAPE 0 1 1 MOD 1 0 1 R/M	DISP	53-65			
BCD Memory to ST(0)	ESCAPE 1 1 1 MOD 1 0 0 R/M	DISP	290-310			
ST(i) to ST(0)	ESCAPE 0 0 1 1 1 0 0 0 ST(i)		17-22			
FST = STORE						
ST(0) to Integer/Real Memory	ESCAPE MF 1 MOD 0 1 0 R/M	DISP	84-90	82-92	96-104	80-90
ST(0) to ST(i)	ESCAPE 1 0 1 1 1 0 1 0 ST(i)		15-22			
FSTP = STORE AND POP						
ST(0) to Integer/Real Memory	ESCAPE MF 1 MOD 0 1 1 R/M	DISP	86-92	84-94	98-106	82-92
ST(0) to Long Integer Memory	ESCAPE 1 1 1 MOD 1 1 1 R/M	DISP	94-105			
ST(0) to Temporary Real Memory	ESCAPE 0 1 1 MOD 1 1 1 R/M	DISP	52-58			
ST(0) to BCD Memory	ESCAPE 1 1 1 MOD 1 1 0 R/M	DISP	520-540			
ST(0) to ST(i)	ESCAPE 1 0 1 1 1 0 1 1 ST(i)		17-24			
FXCH = Exchange ST(i) and ST(0)	ESCAPE 0 0 1 1 1 0 0 1 ST(i)		10-15			
Comparison						
FCOM = Compare						
Integer/Real Memory to ST(0)	ESCAPE MF 0 MOD 0 1 0 R/M	DISP	60-70	78-91	65-75	72-86
ST(i) to ST(0)	ESCAPE 0 0 0 1 1 0 1 0 ST(i)		40-50			
FCOMP = Compare and Pop						
Integer/Real Memory to ST(0)	ESCAPE MF 0 MOD 0 1 1 R/M	DISP	63-73	80-93	67-77	74-88
ST(i) to ST(0)	ESCAPE 0 0 0 1 1 0 1 1 ST(i)		45-52			
FCOMPP = Compare ST(1) to ST(0) and Pop Twice	ESCAPE 1 1 0 1 1 0 1 1 0 0 1		45-55			
FTST = Test ST(0)	ESCAPE 0 0 1 1 1 1 0 0 1 0 0		38-48			
FXAM = Examine ST(0)	ESCAPE 0 0 1 1 1 1 0 0 1 0 1		12-23			



AC Drive and Measurement Points—CLK Input



AC Setup, Hold and Delay Time Measurement—General



AC Test Loading on Outputs

Table 6. 80287 Extensions to the 80286 Instruction Set (Continued)

Constants	MF	=	Optional 8, 16 Bit Displacement	Clock Count Range			
				32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer
				00	01	10	11
FLDZ = LOAD + 0.0 into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 1 1 0	11-17			
FLD1 = LOAD + 1.0 into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 0 0 0	15-21			
FLDPI = LOAD π into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 0 1 1	16-22			
FLDL2T = LOAD $\log_2 10$ into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 0 0 1	16-22			
FLDL2E = LOAD $\log_2 e$ into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 0 1 0	15-21			
FLDLG2 = LOAD $\log_{10} 2$ into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 1 0 0	18-24			
FLDLN2 = LOAD $\log_e 2$ into ST(0)	ESCAPE	0 0 1	1 1 1 0 1 1 0 1	17-23			
Arithmetic							
FADD = Addition							
Integer/Real Memory with ST(0)	ESCAPE	MF 0	MOD 0 0 0 R/M	DISP	90-120	108-143	95-125 102-137
ST(i) and ST(0)	ESCAPE	d P 0	1 1 0 0 0 ST(i)	70-100 (Note 1)			
FSUB = Subtraction							
Integer/Real Memory with ST(0)	ESCAPE	MF 0	MOD 1 0 R R/M	DISP	90-120	108-143	95-125 102-137
ST(i) and ST(0)	ESCAPE	d P 0	1 1 1 0 R R/M	70-100 (Note 1)			
FMUL = Multiplication							
Integer/Real Memory with ST(0)	ESCAPE	MF 0	MOD 0 0 1 R/M	DISP	110-125	130-144	112-168 124-138
ST(i) and ST(0)	ESCAPE	d P 0	1 1 0 0 1 R/M	90-145 (Note 1)			
FDIV = Division							
Integer/Real Memory with ST(0)	ESCAPE	MF 0	MOD 1 1 R R/M	DISP	215-225	230-243	220-230 224-238
ST(i) and ST(0)	ESCAPE	d P 0	1 1 1 1 R R/M	193-203 (Note 1)			
FSQRT = Square Root of ST(0)	ESCAPE	0 0 1	1 1 1 1 1 0 1 0	180-186			
FSCALE = Scale ST(0) by ST(1)	ESCAPE	0 0 1	1 1 1 1 1 1 0 1	32-38			
FPREM = Partial Remainder of ST(0) \div ST(1)	ESCAPE	0 0 1	1 1 1 1 1 0 0 0	15-190			
FRNDINT = Round ST(0) to Integer	ESCAPE	0 0 1	1 1 1 1 1 1 0 0	16-50			

210920-20

NOTE:

1. If P = 1 then add 5 clocks.

Table 6. 80287 Extensions to the 80286 Instruction Set (Continued)

		Optional 8,16 BIT Displacement	Clock Count Range
FEXTRACT = Extract Components of ST(0)	ESCAPE 0 0 1	1 1 1 1 0 1 0 0	27-55
FABS = Absolute Value of ST(0)	ESCAPE 0 0 1	1 1 1 0 0 0 0 1	10-17
FCHS = Change Sign of ST(0)	ESCAPE 0 0 1	1 1 1 0 0 0 0 0	10-17
Transcendental			
FPTAN = Partial Tangent of ST(0)	ESCAPE 0 0 1	1 1 1 1 0 0 1 0	30-540
FPATAN = Partial Arctangent of ST(0) - ST(1)	ESCAPE 0 0 1	1 1 1 1 0 0 1 1	250-800
F2XM1 = $2^{ST(0)} - 1$	ESCAPE 0 0 1	1 1 1 1 0 0 0 0	310-630
FYL2X = $ST(1) \cdot \text{Log}_2 ST(0) $	ESCAPE 0 0 1	1 1 1 1 0 0 0 1	900-1100
FYL2XP1 = $ST(1) \cdot \text{Log}_2 ST(0) + 1 $	ESCAPE 0 0 1	1 1 1 1 1 0 0 1	700-1000
Processor Control			
FINIT = Initialize NPX	ESCAPE 0 1 1	1 1 1 0 0 0 1 1	2-8
FSETPM = Enter Protected Mode	ESCAPE 0 1 1	1 1 1 0 0 1 0 0	2-8
FSTSW AX = Store Control Word	ESCAPE 1 1 1	1 1 1 0 0 0 0 0	10-16
FLDCW = Load Control Word	ESCAPE 0 0 1	MOD 1 0 1 R/M	DISP 7-14
FSTCW = Store Control Word	ESCAPE 0 0 1	MOD 1 1 1 R/M	DISP 12-18
FSTSW = Store Status Word	ESCAPE 1 0 1	MOD 1 1 1 R/M	DISP 12-18
FCLEX = Clear Exceptions	ESCAPE 0 1 1	1 1 1 0 0 0 1 0	2-8
FSTENV = Store Environment	ESCAPE 0 0 1	MOD 1 1 0 R/M	DISP 40-50
FLDENV = Load Environment	ESCAPE 0 0 1	MOD 1 0 0 R/M	DISP 35-45
FSAVE = Save State	ESCAPE 1 0 1	MOD 1 1 0 R/M	DISP 205-215
FRSTOR = Restore State	ESCAPE 1 0 1	MOD 1 0 0 R/M	DISP 205-215
FINCSTP = Increment Stack Pointer	ESCAPE 0 0 1	1 1 1 1 0 1 1 1	6-12
FDECSTP = Decrement Stack Pointer	ESCAPE 0 0 1	1 1 1 1 0 1 1 0	6-12

Table 6. 80287 Extensions to the 80286 Instruction Set (Continued)

		Clock Count Range
FFREE = Free ST(i)	ESCAPE 1 0 1 1 1 0 0 0 ST(i)	9-16
FNOP = No Operation	ESCAPE 0 0 1 1 1 0 1 0 0, 0 0	10-16
		210920-22

NOTES:

- if mod = 00 then DISP = 0*, disp-low and disp-high are absent
 if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
 if mod = 10 then DISP = disp-high; disp-low
 if mod = 11 then r/m is treated as an ST(i) field
- if r/m = 000 then EA = (BX) + (SI) + DISP
 if r/m = 001 then EA = (BX) + (DI) + DISP
 if r/m = 010 then EA = (BP) + (SI) + DISP
 if r/m = 011 then EA = (BP) + (DI) + DISP
 if r/m = 100 then EA = (SI) + DISP
 if r/m = 101 then EA = (DI) + DISP
 if r/m = 110 then EA = (BP) + DISP
 if r/m = 111 then EA = (BX) + DISP
 *except if mod = 000 and r/m = 110 then EA = disp-high; disp-low.
- MF = Memory Format
 00—32-bit Real
 01—32-bit Integer
 10—64-bit Real
 11—16-bit Integer
- ST(0) = Current stack top
 ST(i) = ith register below stack top
- d = Destination
 0—Destination is ST(0)
 1—Destination is ST(i)
- P = Pop
 0—No pop
 1—Pop ST(0)
- R = Reverse: When d = 1 reverse the sense of R
 0—Destination (op) Source
 1—Source (op) Destination
- For **FSQRT**: $-0 \leq ST(0) \leq +\infty$
 For **FSCALE**: $-2^{15} \leq ST(1) < +2^{15}$ and ST(1) integer
 For **F2XM1**: $0 \leq ST(0) \leq 2^{-1}$
 For **FYL2X**: $0 < ST(0) < \infty$
 $-\infty < ST(1) < +\infty$
 For **FYL2XP1**: $0 \leq |ST(0)| < (2 - \sqrt{2})/2$
 $-\infty < ST(1) < \infty$
 For **FPTAN**: $0 \leq ST(0) \leq \pi/4$
 For **FPATAN**: $0 \leq ST(0) < ST(1) < +\infty$
- ESCAPE bit pattern is 11011.



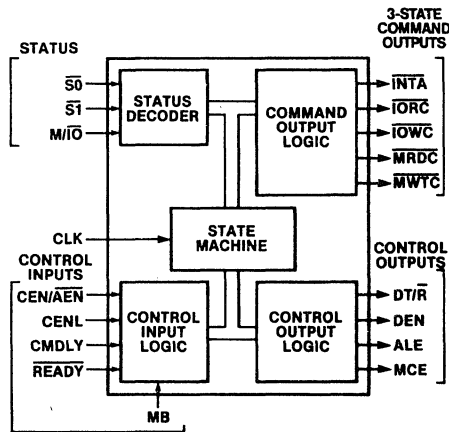
82C288 BUS CONTROLLER FOR 80286 PROCESSORS (82C288-12, 82C288-10, 82C288-8)

- Provides Commands and Controls for Local and System Bus
- Wide Flexibility in System Configurations
- High Speed CHMOS III Technology
- Fully Compatible with the HMOS 82288
- Fully Static Device
- Available in 20 Pin PLCC (Plastic Leaded Chip Carrier) and 20 Pin Cerdip Packages

(See Packaging Spec, Order #231369)

The Intel 82C288 Bus Controller is a 20-pin CHMOS III component for use in 80286 microsystems. The 82C288 is fully compatible with its predecessor the HMOS 82288. The bus controller is fully static and supports a low power mode. The bus controller provides command and control outputs with flexible timing options. Separate command outputs are used for memory and I/O devices. The data bus is controlled with separate data enable and direction control signals.

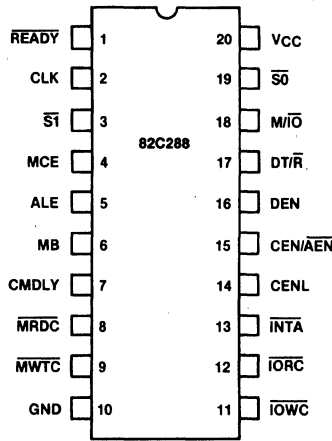
Two modes of operation are possible via a strapping option: MULTIBUS® I compatible bus cycles, and high speed bus cycles.



240042-1

Figure 1. 82C288 Block Diagram

20 Pin Cerdip Package

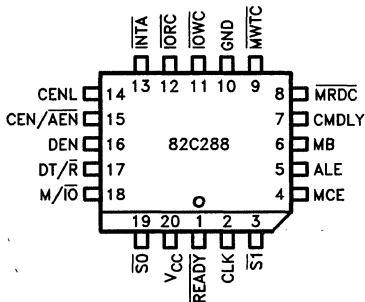


240042-2

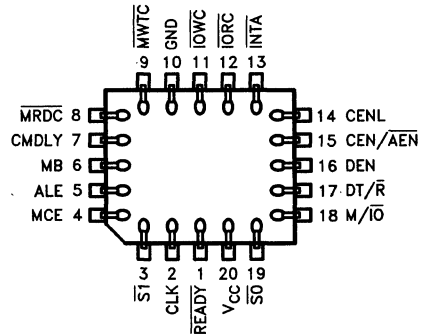
P.C. Board Views—As viewed from the component side of the P.C. board.

Component Pad Views—As viewed from underside of component when mounted on the board.

20 Pin PLCC Package



240042-3



240042-4

Figure 2. 82C288 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for the 82C288 bus controller.

Symbol	Type	Name and Function																																								
CLK	I	SYSTEM CLOCK provides the basic timing control for the 82C288 in an 80286 microsystem. Its frequency is twice the internal processor clock frequency. The falling edge of this input signal establishes when inputs are sampled and command and control outputs change.																																								
$\overline{S0}, \overline{S1}$	I	<p>BUS CYCLE STATUS starts a bus cycle and, along with M/\overline{IO}, defines the type of bus cycle. These inputs are active LOW. A bus cycle is started when either $\overline{S1}$ or $\overline{S0}$ is sampled LOW at the falling edge of CLK. Setup and hold times must be met for proper operation.</p> <table border="1"> <thead> <tr> <th colspan="4">80286 Bus Cycle Status Definition</th> </tr> <tr> <th>M/\overline{IO}</th> <th>$\overline{S1}$</th> <th>$\overline{S0}$</th> <th>Type of Bus Cycle</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>Interrupt Acknowledge</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>I/O Read</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>I/O Write</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>None; Idle</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>Halt or Shutdown</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>Memory Read</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>Memory Write</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>None; Idle</td> </tr> </tbody> </table>	80286 Bus Cycle Status Definition				M/\overline{IO}	$\overline{S1}$	$\overline{S0}$	Type of Bus Cycle	0	0	0	Interrupt Acknowledge	0	0	1	I/O Read	0	1	0	I/O Write	0	1	1	None; Idle	1	0	0	Halt or Shutdown	1	0	1	Memory Read	1	1	0	Memory Write	1	1	1	None; Idle
80286 Bus Cycle Status Definition																																										
M/\overline{IO}	$\overline{S1}$	$\overline{S0}$	Type of Bus Cycle																																							
0	0	0	Interrupt Acknowledge																																							
0	0	1	I/O Read																																							
0	1	0	I/O Write																																							
0	1	1	None; Idle																																							
1	0	0	Halt or Shutdown																																							
1	0	1	Memory Read																																							
1	1	0	Memory Write																																							
1	1	1	None; Idle																																							
M/\overline{IO}	I	MEMORY OR I/O SELECT determines whether the current bus cycle is in the memory space or I/O space. When LOW, the current bus cycle is in the I/O space. Setup and hold times must be met for proper operation.																																								
MB	I	MULTIBUS MODE SELECT determines timing of the command and control outputs. When HIGH, the bus controller operates with MULTIBUS I compatible timings. When LOW, the bus controller optimizes the command and control output timing for short bus cycles. The function of the CEN/\overline{AEN} input pin is selected by this signal. This input is typically a strapping option and not dynamically changed.																																								
CENL	I	COMMAND ENABLE LATCHED is a bus controller select signal which enables the bus controller to respond to the current bus cycle being initiated. CENL is an active HIGH input latched internally at the end of each T_S cycle. CENL is used to select the appropriate bus controller for each bus cycle in a system where the CPU has more than one bus it can use. This input may be connected to V_{CC} to select this 82C288 for all transfers. No control inputs affect CENL. Setup and hold times must be met for proper operation.																																								
CMDLY	I	COMMAND DELAY allows delaying the start of a command. CMDLY is an active HIGH input. If sampled HIGH, the command output is not activated and CMDLY is again sampled at the next CLK cycle. When sampled LOW the selected command is enabled. If \overline{READY} is detected LOW before the command output is activated, the 82C288 will terminate the bus cycle, even if no command was issued. Setup and hold times must be satisfied for proper operation. This input may be connected to GND if no delays are required before starting a command. This input has no effect on 82C288 control outputs.																																								
READY	I	READY indicates the end of the current bus cycle. \overline{READY} is an active LOW input. MULTIBUS I mode requires at least one wait state to allow the command outputs to become active. \overline{READY} must be LOW during reset, to force the 82C288 into the idle state. Setup and hold times must be met for proper operation. The 82C284 drives \overline{READY} LOW during RESET.																																								

Table 1. Pin Description (Continued)

Symbol	Type	Name and Function
CEN/AEN	I	<p>COMMAND ENABLE/ADDRESS ENABLE controls the command and DEN outputs of the bus controller. CEN/AEN inputs may be asynchronous to CLK. Setup and hold times are given to assure a guaranteed response to synchronous inputs. This input may be connected to V_{CC} or GND.</p> <p>When MB is HIGH this pin has the AEN function. AEN is an active LOW input which indicates that the CPU has been granted use of a shared bus and the bus controller command outputs may exit 3-state OFF and become inactive (HIGH). AEN HIGH indicates that the CPU does not have control of the shared bus and forces the command outputs into 3-state OFF and DEN inactive (LOW).</p> <p>When MB is LOW this pin has the CEN function. CEN is an unlatched active HIGH input which allows the bus controller to activate its command and DEN outputs. With MB LOW, CEN LOW forces the command and DEN outputs inactive but does not tristate them.</p>
ALE	O	<p>ADDRESS LATCH ENABLE controls the address latches used to hold an address stable during a bus cycle. This control output is active HIGH. ALE will not be issued for the halt bus cycle and is not affected by any of the control inputs.</p>
MCE	O	<p>MASTER CASCADE ENABLE signals that a cascade address from a master 8259A interrupt controller may be placed onto the CPU address bus for latching by the address latches under ALE control. The CPU's address bus may then be used to broadcast the cascade address to slave interrupt controllers so only one of them will respond to the interrupt acknowledge cycle. This control output is active HIGH. MCE is only active during interrupt acknowledge cycles and is not affected by any control input. Using MCE to enable cascade address drivers requires latches which save the cascade address on the falling edge of ALE.</p>
DEN	O	<p>DATA ENABLE controls when data transceivers connected to the local data bus should be enabled. DEN is an active HIGH control output. DEN is delayed for write cycles in the MULTIBUS I mode.</p>
DT/R	O	<p>DATA TRANSMIT/RECEIVE establishes the direction of data flow to or from the local data bus. When HIGH, this control output indicates that a write bus cycle is being performed. A LOW indicates a read bus cycle. DEN is always inactive when DT/R changes states. This output is HIGH when no bus cycle is active. DT/R is not affected by any of the control inputs.</p>
IOWC	O	<p>I/O WRITE COMMAND instructs an I/O device to read the data on the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. READY controls when it becomes inactive.</p>
IORC	O	<p>I/O READ COMMAND instructs an I/O device to place data onto the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. READY controls when it becomes inactive.</p>
MWTC	O	<p>MEMORY WRITE COMMAND instructs a memory device to read the data on the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. READY controls when it becomes inactive.</p>
MRDC	O	<p>MEMORY READ COMMAND instructs the memory device to place data onto the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. READY controls when it becomes inactive.</p>

Table 1. Pin Description (Continued)

Symbol	Type	Name and Function
$\overline{\text{INTA}}$	0	INTERRUPT ACKNOWLEDGE tells an interrupting device that its interrupt request is being acknowledged. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. READY controls when it becomes inactive.
V_{CC}		System Power: +5V Power Supply
GND		System Ground: 0V

Table 2. Command and Control Outputs for Each Type of Bus Cycle

Type of Bus Cycle	M/\overline{IO}	$\overline{S1}$	$\overline{S0}$	Command Activated	DT/ \overline{R} State	ALE, DEN Issued?	MCE Issued?
Interrupt Acknowledge	0	0	0	$\overline{\text{INTA}}$	LOW	YES	YES
I/O Read	0	0	1	$\overline{IO\overline{RC}}$	LOW	YES	NO
I/O Write	0	1	0	$\overline{IO\overline{WC}}$	HIGH	YES	NO
None; Idle	0	1	1	None	HIGH	NO	NO
Halt/Shutdown	1	0	0	None	HIGH	NO	NO
Memory Read	1	0	1	\overline{MRDC}	LOW	YES	NO
Memory Write	1	1	0	\overline{MWTC}	HIGH	YES	NO
None; Idle	1	1	1	None	HIGH	NO	NO

Operating Modes

Two types of buses are supported by the 82C288: MULTIBUS I and non-MULTIBUS I. When the MB input is strapped HIGH, MULTIBUS I timing is used. In MULTIBUS I mode, the 82C288 delays command and data activation to meet IEEE-796 requirements on address to command active and write data to command active setup timing. MULTIBUS I mode requires at least one wait state in the bus cycle since the command outputs are delayed. The non-MULTIBUS I mode does not delay any outputs and does not require wait states. The MB input affects the timing of the command and DEN outputs.

Command and Control Outputs

The type of bus cycle performed by the local bus master is encoded in the M/\overline{IO} , $\overline{S1}$, and $\overline{S0}$ inputs. Different command and control outputs are activated depending on the type of bus cycle. Table 2 indicates the cycle decode done by the 82C288 and the effect on command, DT/ \overline{R} , ALE, DEN, and MCE outputs.

Bus cycles come in three forms: read, write, and halt. Read bus cycles include memory read, I/O read, and interrupt acknowledge. The timing of the associated read command outputs (\overline{MRDC} , $\overline{IO\overline{RC}}$,

and $\overline{\text{INTA}}$), control outputs (ALE, DEN, DT/ \overline{R}) and control inputs (CEN/ $\overline{\text{AEN}}$, CENL, CMDLY, MB, and **READY**) are identical for all read bus cycles. Read cycles differ only in which command output is activated. The MCE control output is only asserted during interrupt acknowledge cycles.

Write bus cycles activate different control and command outputs with different timing than read bus cycles. Memory write and I/O write are write bus cycles whose timing for command outputs (\overline{MWTC} and $\overline{IO\overline{WC}}$), control outputs (ALE, DEN, DT/ \overline{R}) and control inputs (CEN/ $\overline{\text{AEN}}$, CENL, CMDLY, MB, and **READY**) are identical. They differ only in which command output is activated.

Halt bus cycles are different because no command or control output is activated. All control inputs are ignored until the next bus cycle is started via $\overline{S1}$ and $\overline{S0}$.

Static Operation

All 82C288 circuitry is of static design. Internal registers and logic are static and require no refresh as with dynamic circuit design. This eliminates the minimum operating frequency restriction placed on the HMOS 82288. The CHMOS III 82C288 can operate from DC to the appropriate upper frequency limit.

The clock may be stopped in either state (HIGH/LOW) and held there indefinitely.

Power dissipation is directly related to operating frequency. As the system frequency is reduced, so is the operating power. When the clock is stopped to the 82C288, power dissipation is at a minimum. This is useful for low-power and portable applications.

FUNCTIONAL DESCRIPTION

Introduction

The 82C288 bus controller is used in 80286 systems to provide address latch control, data transceiver control, and standard level-type command outputs. The command outputs are timed and have sufficient drive capabilities for large TTL buses and meet all IEEE-796 requirements for MULTIBUS I. A special MULTIBUS I mode is provided to satisfy all address/data setup and hold time requirements. Command timing may be tailored to special needs via a CMDLY input to determine the start of a command and READY to determine the end of a command.

Connection to multiple buses are supported with a latched enable input (CENL). An address decoder can determine which, if any, bus controller should be enabled for the bus cycle. This input is latched to allow an address decoder to take full advantage of the pipelined timing on the 80286 local bus.

Buses shared by several bus controllers are supported. An \overline{AEN} input prevents the bus controller from driving the shared bus command and data signals except when enabled by an external MULTIBUS I type bus arbiter.

Separate DEN and DT/ \overline{R} outputs control the data transceivers for all buses. Bus contention is eliminated by disabling DEN before changing DT/ \overline{R} . The DEN timing allows sufficient time for tristate bus drivers to enter 3-state OFF before enabling other drivers onto the same bus.

The term CPU refers to any 80286 processor or 80286 support component which may become an 80286 local bus master and thereby drive the 82C288 status inputs.

Processor Cycle Definition

Any CPU which drives the local bus uses an internal clock which is one half the frequency of the system clock (CLK) (see Figure 3). Knowledge of the phase of the local bus master internal clock is required for proper operation of the 80286 local bus. The local bus master informs the bus controller of its internal clock phase when it asserts the status signals. Status signals are always asserted beginning in Phase 1 of the local bus master's internal clock.

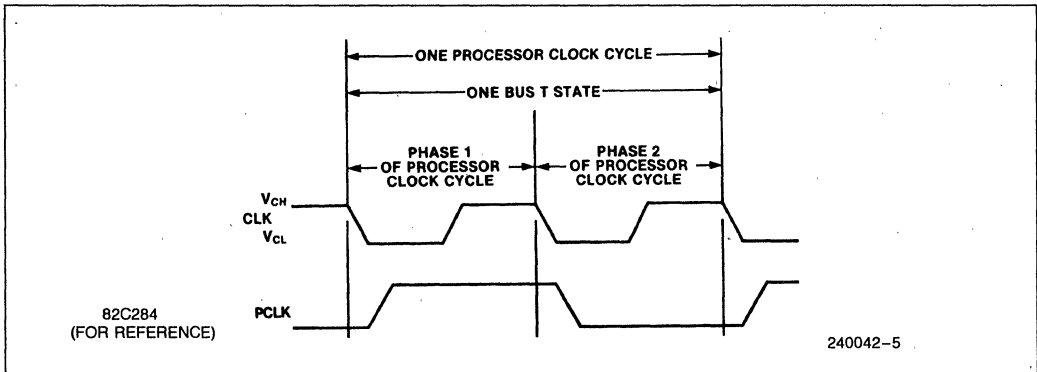


Figure 3. CLK Relationship to the Processor Clock and Bus T-States

Bus State Definition

The 82C288 bus controller has three bus states (see Figure 4): Idle (T_I) Status (T_S) and Command (T_C). Each bus state is two CLK cycles long. Bus state phases correspond to the internal CPU processor clock phases.

The T_I bus state occurs when no bus cycle is currently active on the 80286 local bus. This state may be repeated indefinitely. When control of the local bus is being passed between masters, the bus remains in the T_I state.

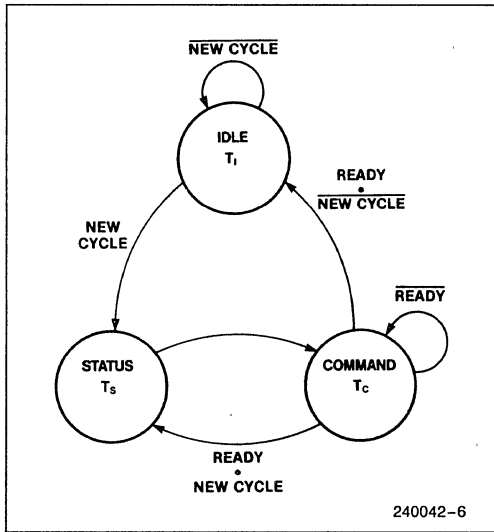


Figure 4. 82C288 Bus States

Bus Cycle Definition

The $\overline{S1}$ and $\overline{S0}$ inputs signal the start of a bus cycle. When either input becomes LOW, a bus cycle is started. The T_S bus state is defined to be the two CLK cycles during which either $\overline{S1}$ or $\overline{S0}$ are active (see Figure 5). These inputs are sampled by the 82C288 at every falling edge of CLK. When either $\overline{S1}$ or $\overline{S0}$ are sampled LOW, the next CLK cycle is considered the second phase of the internal CPU clock cycle.

The local bus enters the T_C bus state after the T_S state. The shortest bus cycle may have one T_S state and one T_C state. Longer bus cycles are formed by repeating T_C state. A repeated T_C bus state is called a wait state.

The \overline{READY} input determines whether the current T_C bus state is to be repeated. The \overline{READY} input has the same timing and effect for all bus cycles. \overline{READY} is sampled at the end of each T_C bus state to see if it is active. If sampled HIGH, the T_C bus state is repeated. This is called inserting a wait state. The control and command outputs do not change during wait states.

When \overline{READY} is sampled LOW, the current bus cycle is terminated. Note that the bus controller may enter the T_S bus state directly from T_C if the status lines are sampled active at the next falling edge of CLK.

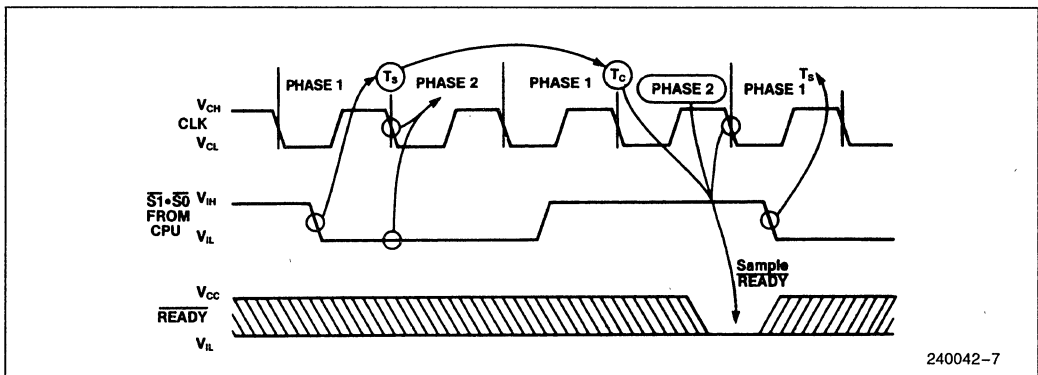


Figure 5. Bus Cycle Definition

Figures 6 through 10 show the basic command and control output timing for read and write bus cycles. Halt bus cycles are not shown since they activate no outputs. The basic idle-read-idle and idle-write-idle bus cycles are shown. The signal label CMD represents the appropriate command output for the bus cycle. For Figures 6 through 10, the CMDLY input is connected to GND and CENL to V_{CC} . The effects of CENL and CMDLY are described later in the section on control inputs.

Figures 6, 7 and 8 show non-MULTIBUS I cycles. MB is connected to GND while CEN is connected to V_{CC} . Figure 6 shows a read cycle with no wait states while Figure 7 shows a write cycle with one wait state. The $\overline{\text{READY}}$ input is shown to illustrate how wait states are added.

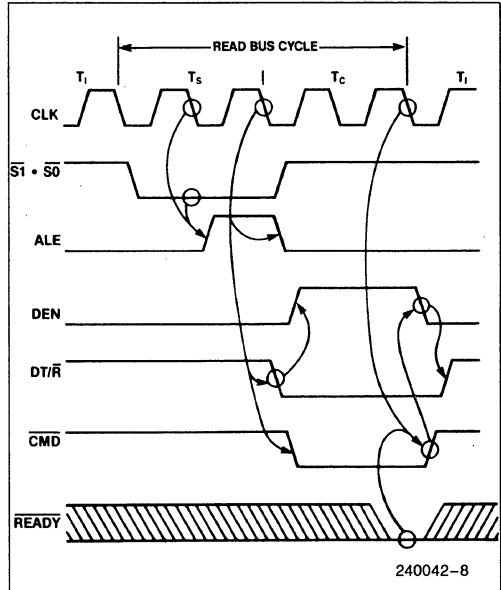


Figure 6. Idle-Read-Idle Bus Cycles with MB = 0

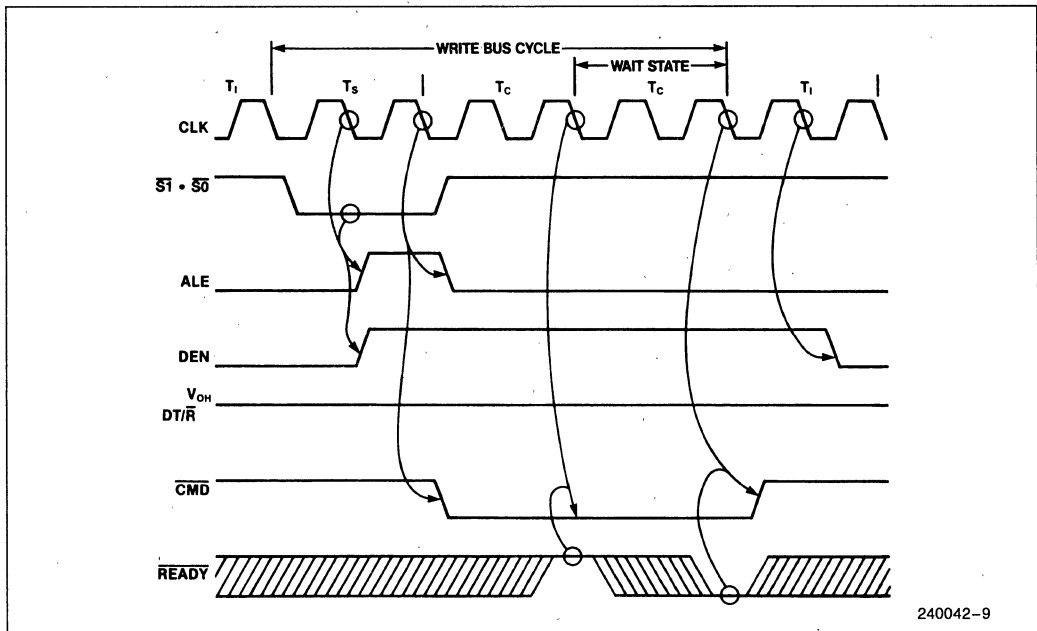


Figure 7. Idle-Write-Idle Bus Cycles with MB = 0

Bus cycles can occur back to back with no T_1 bus states between T_C and T_S . Back to back cycles do not affect the timing of the command and control outputs. Command and control outputs always reach the states shown for the same clock edge (within T_S , T_C or following bus state) of a bus cycle.

A special case in control timing occurs for back to back write cycles with $MB = 0$. In this case, DT/\bar{R} and DEN remain HIGH between the bus cycles (see Figure 8). The command and ALE output timing does not change.

Figures 9 and 10 show a MULTIBUS I cycle with $MB = 1$. \bar{AEN} and $CMDLY$ are connected to GND. The effects of $CMDLY$ and \bar{AEN} are described later in the section on control inputs. Figure 9 shows a read cycle with one wait state and Figure 10 shows a write cycle with two wait states. The second wait state of the write cycle is shown only for example purposes and is not required. The \bar{READY} input is shown to illustrate how wait states are added.

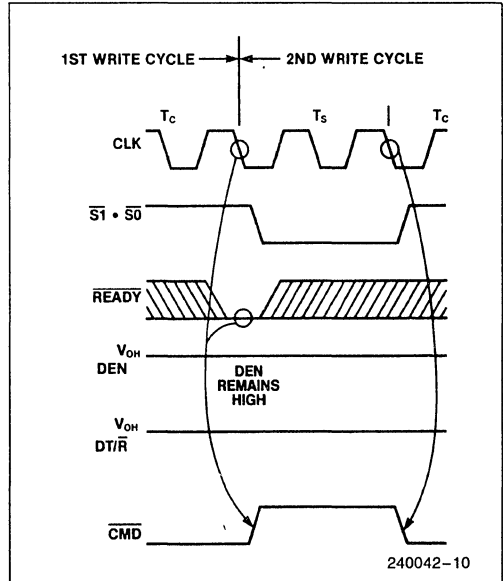


Figure 8. Write-Write Bus Cycles with $MB = 0$

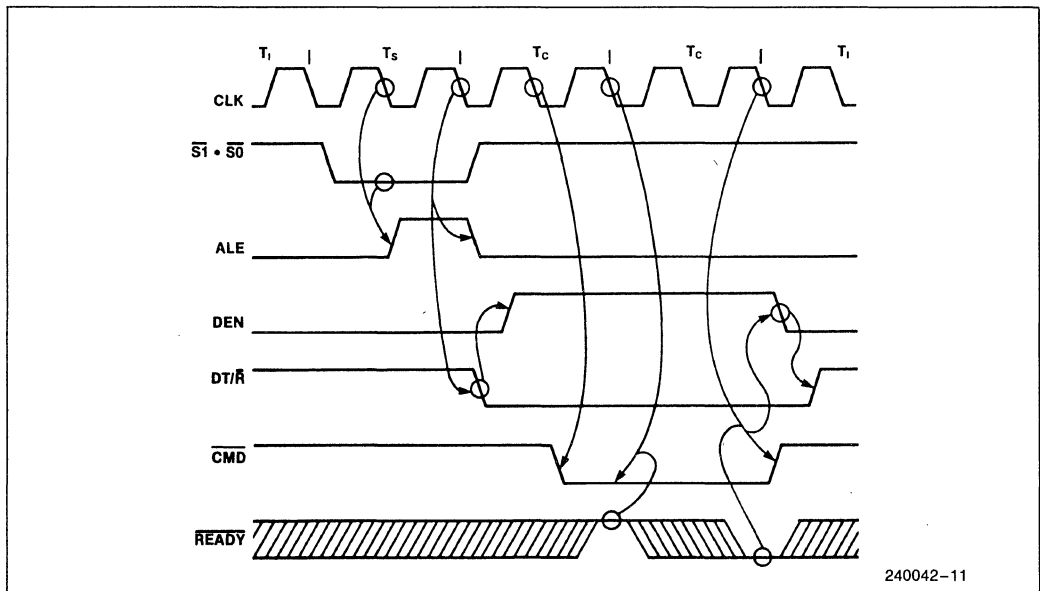


Figure 9. Idle-Read-Idle Bus Cycles with 1 Wait State and with $MB = 1$

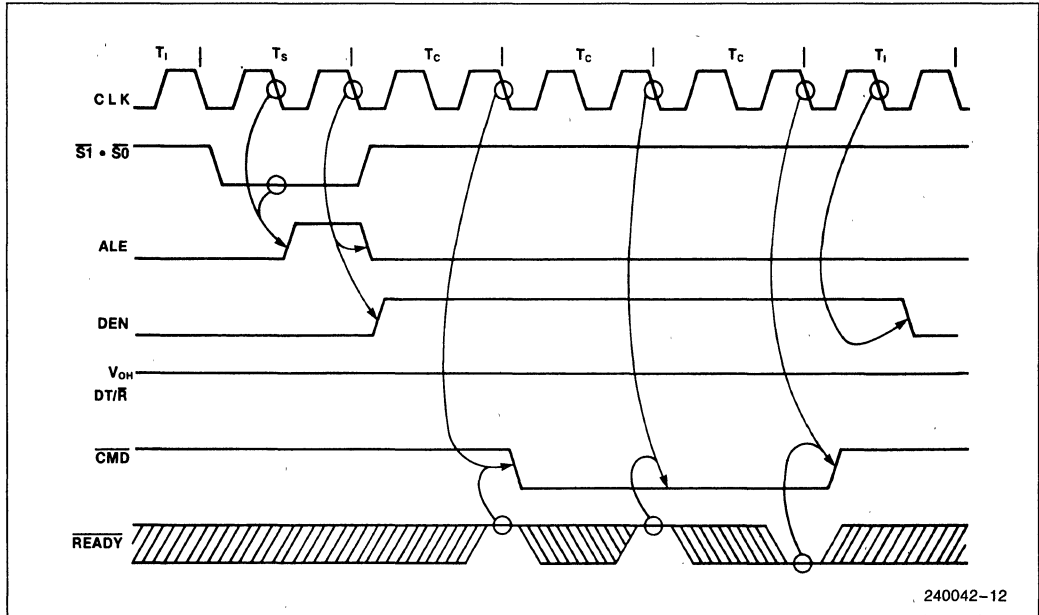


Figure 10. Idle-Write-Idle Bus Cycles with 2 Wait States and with MB = 1

The MB control input affects the timing of the command and DEN outputs. These outputs are automatically delayed in MULTIBUS I mode to satisfy three requirements:

- 1) 50 ns minimum setup time for valid address before any command output becomes active.
- 2) 50 ns minimum setup time for valid write data before any write command output becomes active.
- 3) 65 ns maximum time from when any read command becomes inactive until the slave's read data drivers reach 3-state OFF.

Three signal transitions are delayed by MB = 1 as compared to MB = 0:

- 1) The HIGH to LOW transition of the read command outputs (IORC, MRDC, and INTA) are delayed one CLK cycle.
- 2) The HIGH to LOW transition of the write command outputs (IOWC and MWTC) are delayed two CLK cycles.
- 3) The LOW to HIGH transition of DEN for write cycles is delayed one CLK cycle.

Back to back bus cycles with MB = 1 do not change the timing of any of the command or control outputs. DEN always becomes inactive between bus cycles with MB = 1.

Except for a halt or shutdown bus cycle, ALE will be issued during the second half of Ts for any bus cycle. ALE becomes inactive at the end of the Ts to allow latching the address to keep it stable during the entire bus cycle. The address outputs may change during Phase 2 of any Tc bus state. ALE is not affected by any control input.

Figure 11 shows how MCE is timed during interrupt acknowledge (INTA) bus cycles. MCE is one CLK cycle longer than ALE to hold the cascade address from a master 8259A valid after the falling edge of ALE. With the exception of the MCE control output, an INTA bus cycle is identical in timing to a read bus cycle. MCE is not affected by any control input.

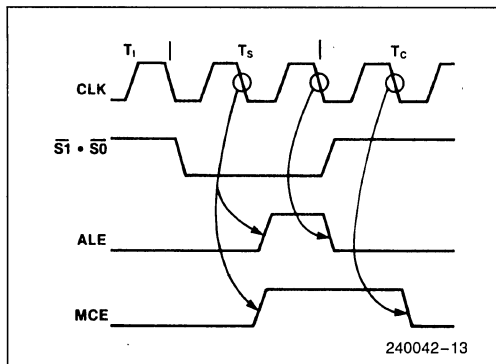


Figure 11. MCE Operation for an INTA Bus Cycle

Control Inputs

The control inputs can alter the basic timing of command outputs, allow interfacing to multiple buses, and share a bus between different masters. For many 80286 systems, each CPU will have more than one bus which may be used to perform a bus cycle. Normally, a CPU will only have one bus controller active for each bus cycle. Some buses may be shared by more than one CPU (i.e. MULTIBUS) requiring only one of them use the bus at a time.

Systems with multiple and shared buses use two control input signals of the 82C288 bus controller, CENL and AEN (see Figure 12). CENL enables the bus controller to control the current bus cycle. The AEN input prevents a bus controller from driving its command outputs. AEN HIGH means that another bus controller may be driving the shared bus.

In Figure 12, two buses are shown: a local bus and a MULTIBUS I. Only one bus is used for each CPU bus cycle. The CENL inputs of the bus controller select which bus controller is to perform the bus cycle. An address decoder determines which bus to use for each bus cycle. The 82C288 connected to the shared MULTIBUS I must be selected by CENL and be given access to the MULTIBUS I by AEN before it will begin a MULTIBUS I operation.

CENL must be sampled HIGH at the end of the T_S bus state (see waveforms) to enable the bus controller to activate its command and control outputs. If sampled LOW the commands and DEN will not go active and DT/R will remain HIGH. The bus controller will ignore the CMDLY, CEN, and READY inputs until another bus cycle is started via $S\bar{1}$ and $S\bar{0}$. Since an address decoder is commonly used to identify which bus is required for each bus cycle, CENL is latched to avoid the need for latching its input.

The CENL input can affect the DEN control output. When $MB = 0$, DEN normally becomes active during Phase 2 of T_S in write bus cycles. This transition occurs before CENL is sampled. If CENL is sampled LOW, the DEN output will be forced LOW during T_C as shown in the timing waveforms.

When $MB = 1$, CEN/\overline{AEN} becomes \overline{AEN} . \overline{AEN} controls when the bus controller command outputs enter and exit 3-state OFF. \overline{AEN} is intended to be driven by a MULTIBUS I type bus arbiter, which assures only one bus controller is driving the shared bus at any time. When AEN makes a LOW to HIGH transition, the command outputs immediately enter 3-state OFF and DEN is forced inactive. An inactive DEN should force the local data transceivers connected to the shared data bus into 3-state OFF (see Figure 12). The LOW to HIGH transition of \overline{AEN} should only occur during T_1 or T_S bus states.

The HIGH to LOW transition of \overline{AEN} signals that the bus controller may now drive the shared bus command signals. Since a bus cycle may be active or be in the process of starting, \overline{AEN} can become active during any T-state. AEN LOW immediately allows DEN to go to the appropriate state. Three CLK edges later, the command outputs will go active (see timing waveforms). The MULTIBUS I requires this delay for the address and data to be valid on the bus before the command becomes active.

When $MB = 0$, CEN/\overline{AEN} becomes CEN. CEN is an asynchronous input which immediately affects the command and DEN outputs. When CEN makes a HIGH to LOW transition, the commands and DEN

are immediately forced inactive. When CEN makes a LOW to HIGH transition, the commands and DEN outputs immediately go to the appropriate state (see timing waveforms). READY must still become active to terminate a bus cycle if CEN remains LOW for a selected bus controller (CENL was latched HIGH).

Some memory or I/O systems may require more address or write data setup time to command active than provided by the basic command output timing. To provide flexible command timing, the CMDLY input can delay the activation of command outputs. The CMDLY input must be sampled LOW to activate the command outputs. CMDLY does not affect the control outputs ALE, MCE, DEN, and DT/R.

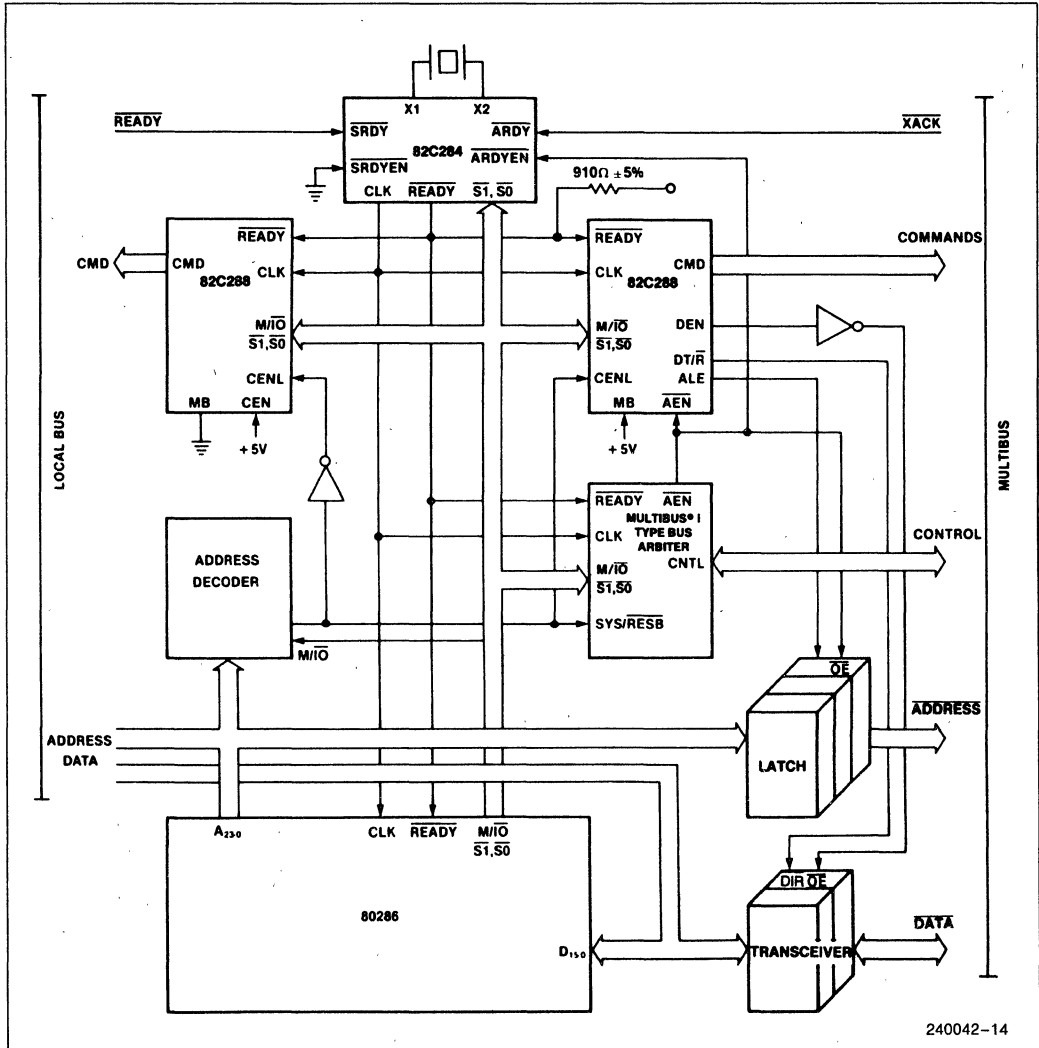


Figure 12. System Use of AEN and CENL

CMDLY is first sampled on the falling edge of the CLK ending T_S . If sampled HIGH, the command output is not activated, and CMDLY is again sampled on the next falling edge of CLK. Once sampled LOW, the proper command output becomes active immediately if $MB = 0$. If $MB = 1$, the proper command goes active no earlier than shown in Figures 9 and 10.

\overline{READY} can terminate a bus cycle before CMDLY allows a command to be issued. In this case no commands are issued and the bus controller will deactivate DEN and DT/\overline{R} in the same manner as if a command had been issued.

Waveforms Discussion

The waveforms show the timing relationships of inputs and outputs and do not show all possible tran-

sitions of all signals in all modes. Instead, all signal timing relationships are shown via the general cases. Special cases are shown when needed. The waveforms provide some functional descriptions of the 82C288; however, most functional descriptions are provided in Figures 5 through 11.

To find the timing specification for a signal transition in a particular mode, first look for a special case in the waveforms. If no special case applies, then use a timing specification for the same or related function in another mode.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias	0°C to +70°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin with Respect to GND	-0.5V to +7V
Power Dissipation	1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $V_{CC} = 5V \pm 5\%$, $T_{CASE} = 0^\circ C$ to $85^\circ C^*$

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input LOW Voltage	-0.5	0.8	V	
V_{IH}	Input HIGH Voltage	2.0	$V_{CC} + 0.5$	V	
V_{ILC}	CLK Input LOW Voltage	-0.5	0.6	V	
V_{IHC}	CLK Input HIGH Voltage	3.8	$V_{CC} + 0.5$	V	
V_{OL}	Output LOW Voltage Command Outputs		0.45	V	$I_{OL} = 32$ mA (Note 1) $I_{OL} = 16$ mA (Note 2)
	Control Outputs		0.45	V	
V_{OH}	Output HIGH Voltage Command Outputs	2.4		V	$I_{OH} = -5$ mA (Note 1) $I_{OH} = -1$ mA (Note 1) $I_{OH} = -1$ mA (Note 2) $I_{OH} = -0.2$ mA (Note 2)
		$V_{CC} - 0.5$		V	
	Control Outputs	2.4		V	
		$V_{CC} - 0.5$		V	
I_{IL}	Input Leakage Current		± 10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		± 10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
I_{CC}	Power Supply Current		75	mA	
I_{CCS}	Power Supply Current (Static)		3	mA	(Note 3)
C_{CLK}	CLK Input Capacitance		12	pF	$F_C = 1$ MHz
C_I	Input Capacitance		10	pF	$F_C = 1$ MHz
C_O	Input/Output Capacitance		20	pF	$F_C = 1$ MHz

* T_A is guaranteed from 0°C to +70°C as long as T_{CASE} is not exceeded.

NOTES:

1. Command Outputs are \overline{INTA} , \overline{IORC} , \overline{IOWC} , \overline{MRDC} and \overline{MWRC} .
2. Control Outputs are DT/\overline{R} , DEN , ALE and MCE .
3. Tested while outputs are unloaded, and inputs at V_{CC} or V_{SS} .

A.C. CHARACTERISTICS

$V_{CC} = 5V, \pm 5\%$, $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$. * AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in data sheet waveforms, unless otherwise noted.

Symbol	Parameter	8 MHz		10 MHz		12.5 MHz		Unit	Test Condition
		-8 Min	-8 Max	-10 Min	-10 Max	-12 Min	-12 Max		
1	CLK Period	62	250	50	250	40	250	ns	
2	CLK HIGH Time	20		16		13		ns	at 3.6V
3	CLK LOW Time	15		12		11		ns	at 1.0V
4	CLK Rise Time		10		8		8	ns	1.0V to 3.6V
5	CLK Fall Time		10		8		8	ns	3.6V to 1.0V
6	M/ \overline{IO} and Status Setup Time	22		18		15		ns	
7	M/ \overline{IO} and Status Hold Time	1		1		1		ns	
8	CENL Setup Time	20		15		15		ns	
9	CENL Hold Time	1		1		1		ns	
10	\overline{READY} Setup Time	38		26		18		ns	
11	\overline{READY} Hold Time	25		25		20		ns	
12	CMDLY Setup Time	20		15		15		ns	
13	CMDLY Hold Time	1		1		1		ns	
14	\overline{AEN} Setup Time	20		15		15		ns	(Note 3)
15	\overline{AEN} Hold Time	0		0		0		ns	(Note 3)
16	ALE, MCE Active Delay from CLK	3	20	3	16	3	16	ns	(Note 4)
17	ALE, MCE Inactive Delay from CLK		25		19		19	ns	(Note 4)
18	DEN (Write) Inactive from CENL		35		23		23	ns	(Note 4)
19	DT/ \overline{R} LOW from CLK		25		23		23	ns	(Note 4)
20	DEN (Read) Active \overline{R} from DT/	5	35	5	21	5	21	ns	(Note 4)
21	DEN (Read) Inactive Dly from CLK	3	35	3	21	3	19	ns	(Note 4)
22	DT/ \overline{R} HIGH from DEN Inactive	5	35	5	20	5	18	ns	(Note 4)
23	DEN (Write) Active Delay from CLK		30		23		23	ns	(Note 4)
24	DEN (Write) Inactive Dly from CLK	3	30	3	19	3	19	ns	(Note 4)

* T_A is guaranteed from $0^{\circ}C$ to $+70^{\circ}C$ as long as T_{CASE} is not exceeded.

A.C. CHARACTERISTICS

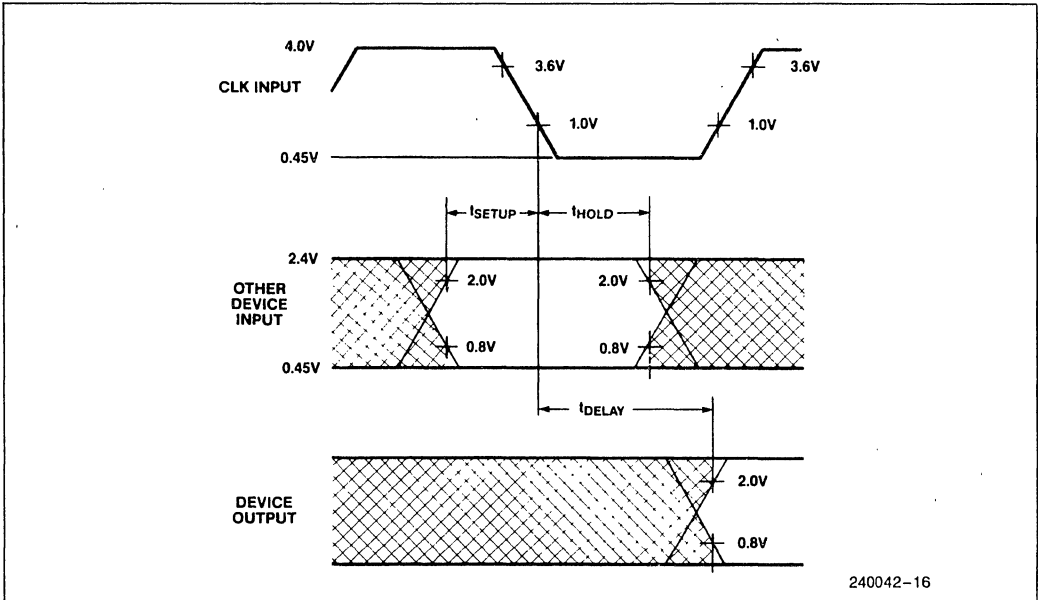
$V_{CC} = 5V, \pm 5\%$, $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$. * AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in data sheet waveforms, unless otherwise noted. (Continued)

Symbol	Parameter	8 MHz		10 MHz		12.5 MHz		Unit	Test Condition
		-8 Min	-8 Max	-10 Min	-10 Max	-12 Min	-12 Max		
25	DEN Inactive from CEN		30		25		25	ns	(Note 4)
26	DEN Active from CEN		30		24		24	ns	(Note 4)
27	DT/ \bar{R} HIGH from CLK (when CEN = LOW)		35		25		25	ns	(Note 4)
28	DEN Active from \bar{AEN}		30		26		26	ns	(Note 4)
29	\overline{CMD} Active Delay from CLK	3	25	3	21	3	21	ns	(Note 5)
30	\overline{CMD} Inactive Delay from CLK	5	20	5	20	5	20	ns	(Note 5)
31	\overline{CMD} Active from CEN		25		25		25	ns	(Note 5)
32	\overline{CMD} Inactive from CEN		25		25		25	ns	(Note 5)
33	\overline{CMD} Inactive Enable from \bar{AEN}		40		40		40	ns	(Note 5)
34	\overline{CMD} Float Delay from \bar{AEN}		40		40		40	ns	(Note 6)
35	MB Setup Time	20		20		20		ns	
36	MB Hold Time	0		0		0		ns	
37	Command Inactive Enable from MB ↓		40		40		40	ns	(Note 5)
38	Command Float Time from MB ↑		40		40		40	ns	(Note 6)
39	DEN Inactive from MB ↑		30		26		26	ns	(Note 4)
40	DEN Active from MB ↓		30		30		30	ns	(Note 4)

* T_A is guaranteed from $0^{\circ}C$ to $+70^{\circ}C$ as long as T_{CASE} is not exceeded.

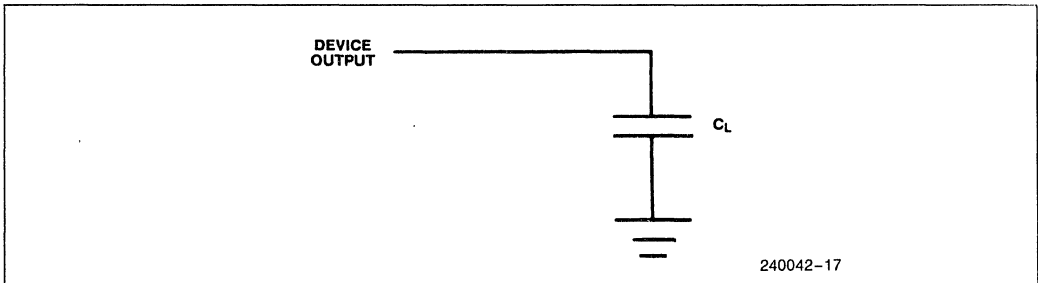
NOTES:

- \bar{AEN} is an asynchronous input. This specification is for testing purposes only, to assure recognition at a specific CLK edge.
- Control output load: $C_I = 150$ pF.
- Command output load: $C_I = 300$ pF.
- Float condition occurs when output current is less than I_{LO} in magnitude.



240042-16

Note 7: AC Setup, Hold and Delay Time Measurement—General

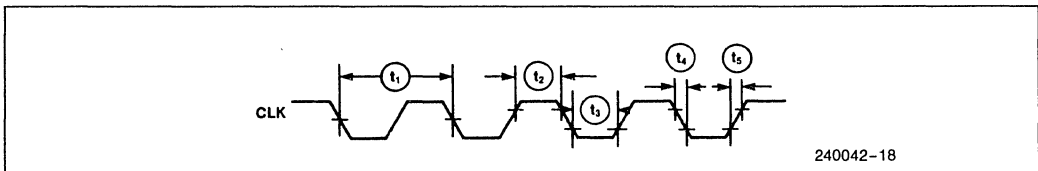


240042-17

Note 8: AC Test Loading on Outputs

WAVEFORMS

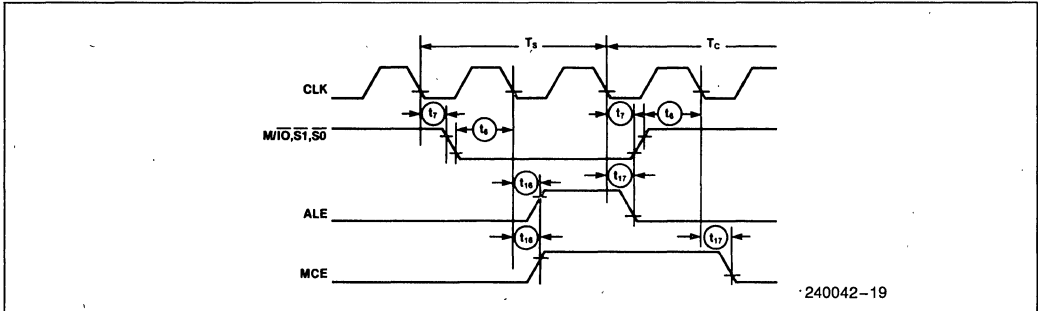
CLK CHARACTERISTICS



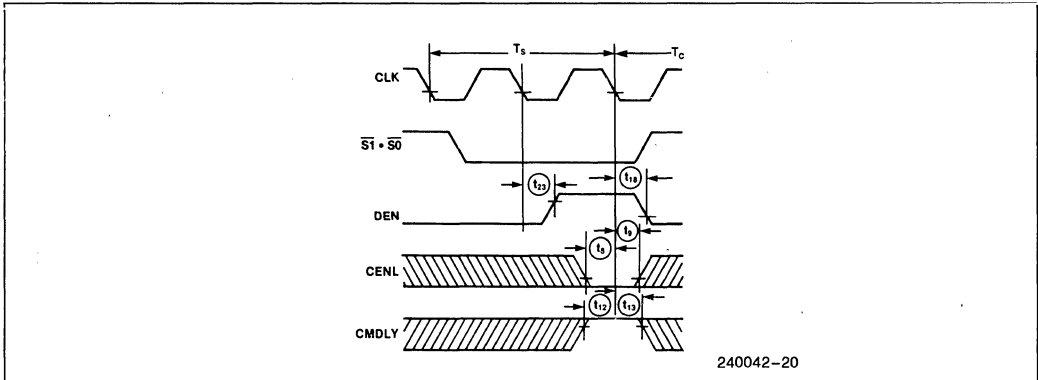
240042-18

WAVEFORMS (Continued)

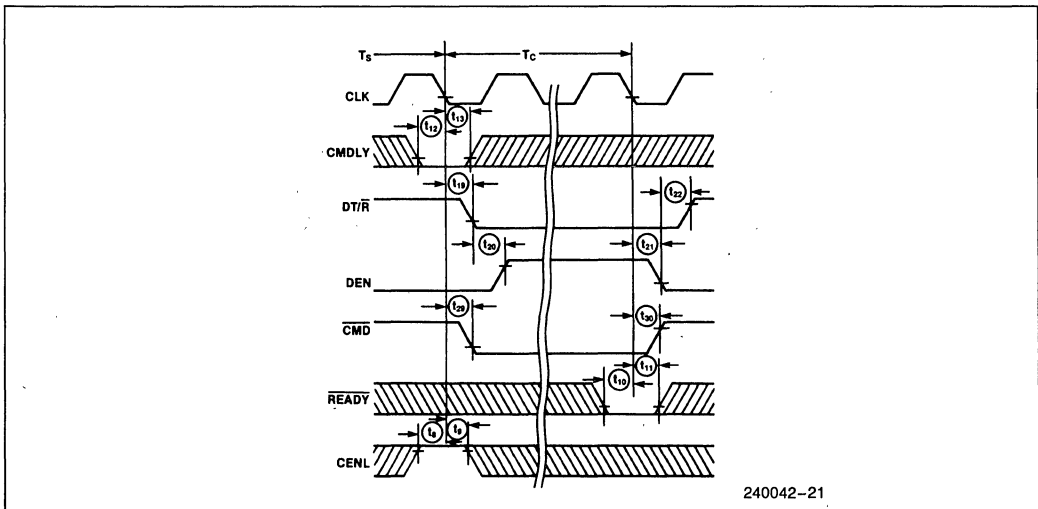
STATUS, ALE, MCE, CHARACTERISTICS



CENL, CMDLY, DEN CHARACTERISTICS WITH MB = 0 AND CEN = 1 DURING WRITE CYCLE

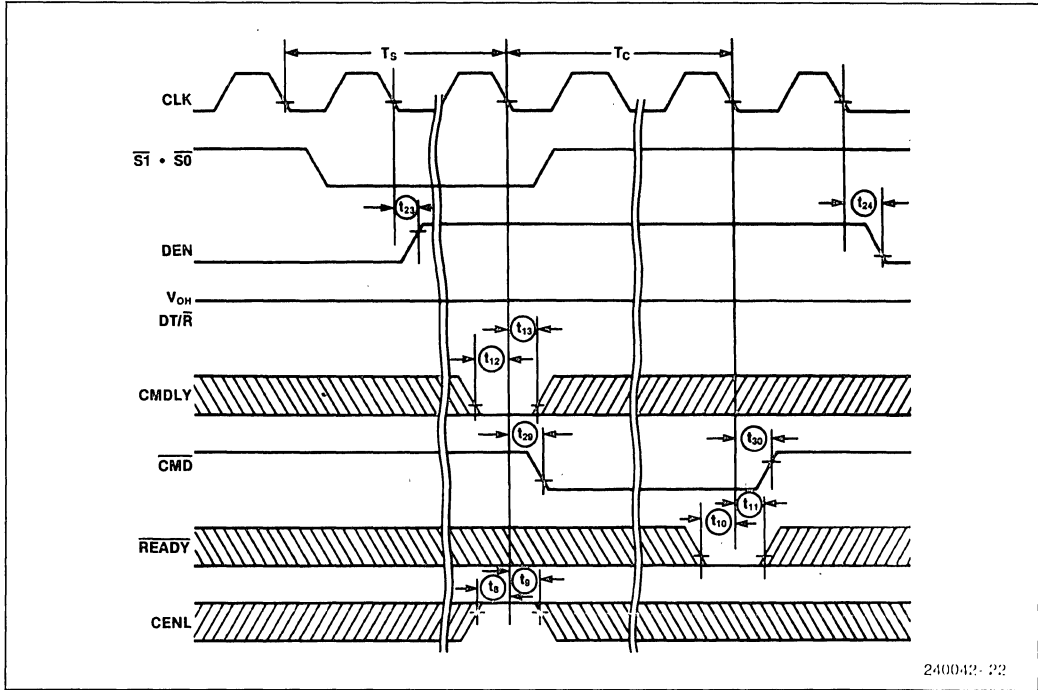


READ CYCLE CHARACTERISTICS WITH MB = 0 AND CEN = 1



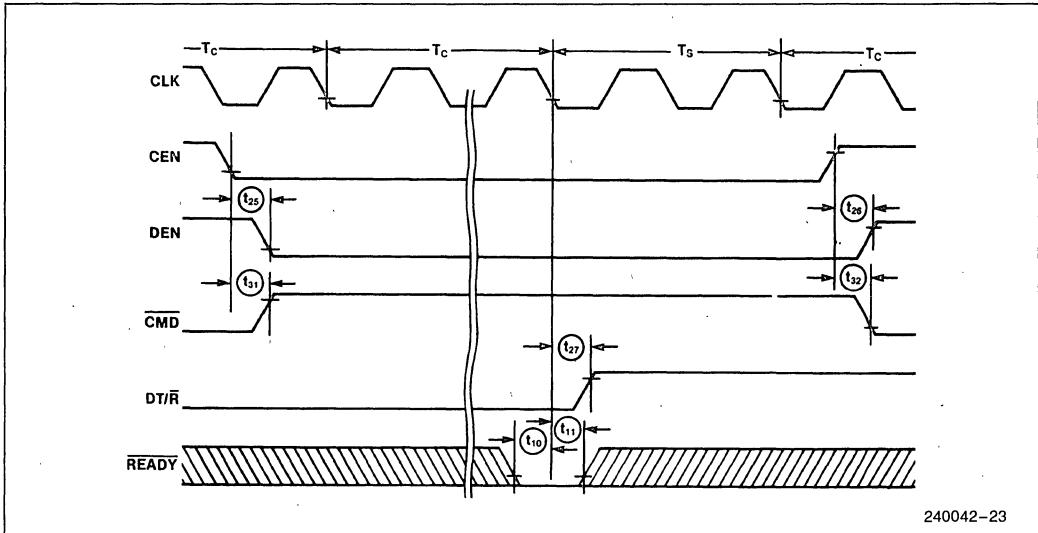
WAVEFORMS (Continued)

WRITE CYCLE CHARACTERISTIC WITH MB = 0 AND CEN = 1



240042-22

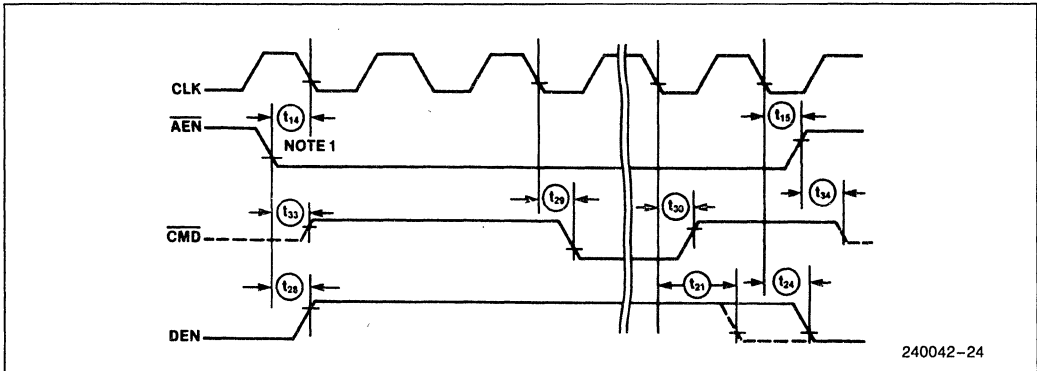
CEN CHARACTERISTICS WITH MB = 0



240042-23

WAVEFORMS (Continued)

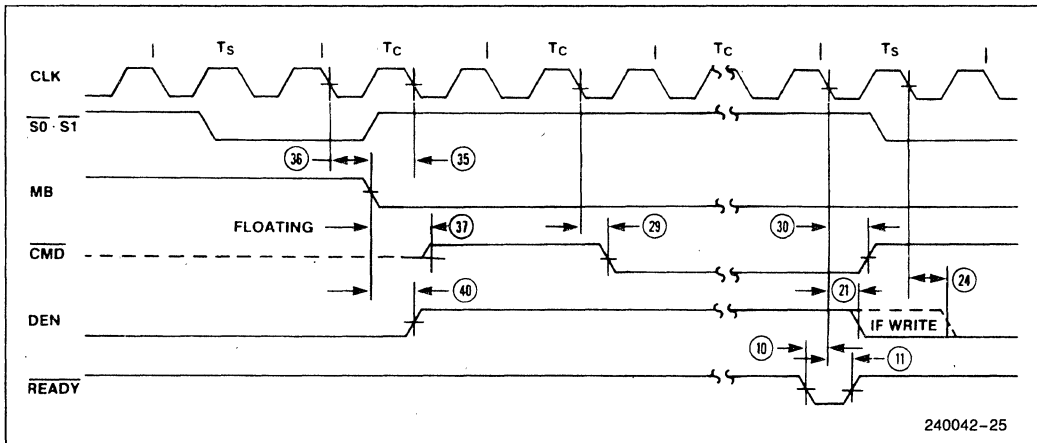
$\overline{\text{AEN}}$ CHARACTERISTICS WITH $\text{MB} = 1$



NOTE:

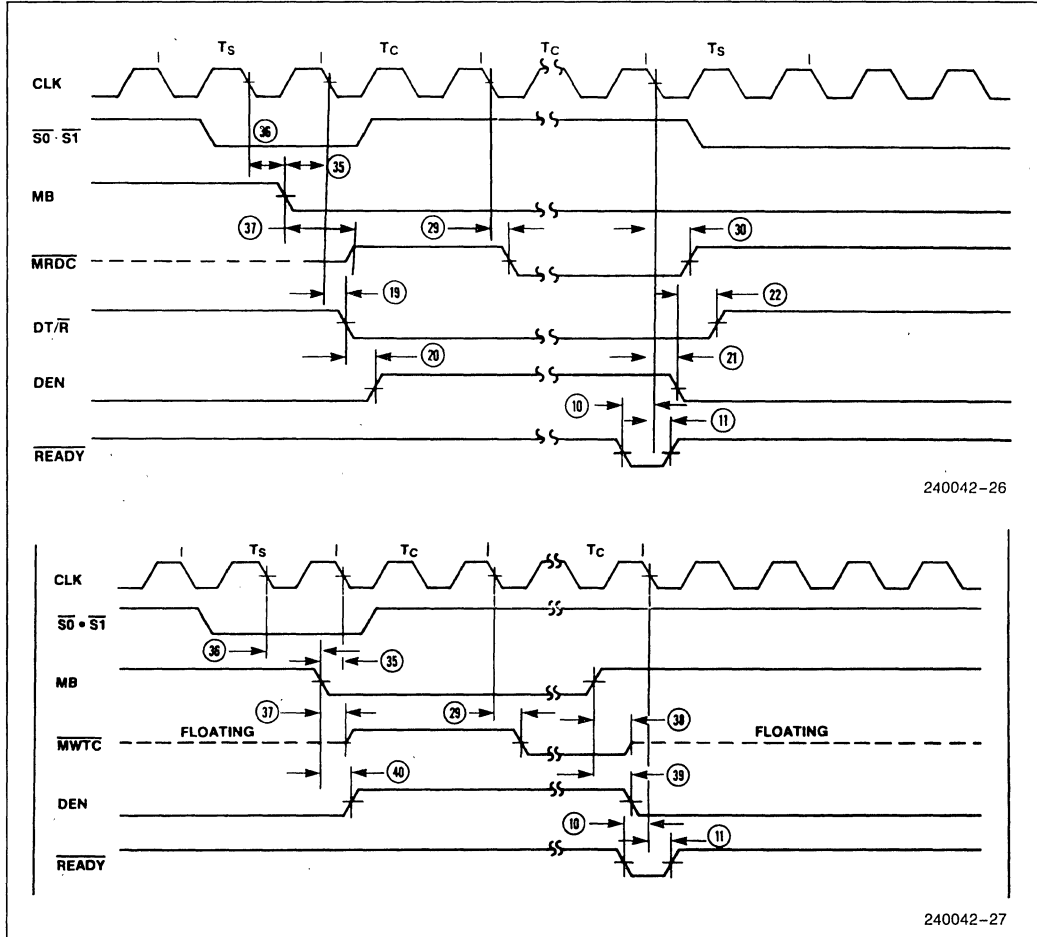
1. $\overline{\text{AEN}}$ is an asynchronous input. $\overline{\text{AEN}}$ setup and hold time is specified to guarantee the response shown in the waveforms.

MB CHARACTERISTICS WITH $\overline{\text{AEN/CEN}} = \text{HIGH}$



WAVEFORMS (Continued)

MB CHARACTERISTICS WITH $\overline{\text{AEN/CEN}} = \text{HIGH}$ (Continued)



NOTES:

1. MB is an asynchronous input. MB setup and hold times specified to guarantee the response shown in the waveforms.
2. If the setup time, t35, is met two clock cycles will occur before CMD becomes active after the falling edge of MB.

DATA SHEET REVISION REVIEW

The following list represents key differences between this and the -002 data sheet. Please review this summary carefully.

1. The ICCS specification was changed from 1 mA to 3 mA maximum.
2. The "PRELIMINARY" markings have been removed from the data sheet.



82C284 CLOCK GENERATOR AND READY INTERFACE FOR 80286 PROCESSORS (82C284-12, 82C284-10, 82C284-8)

- Generates System Clock for 80286 Processors
 - Uses Crystal or TTL Signal for Frequency Source
 - Provides Local READY and MULTIBUS® I READY Synchronization
 - High Speed CHMOS III Technology
 - Generates System Reset Output
 - Available in 18-Lead Cerdip and 20-Pin PLCC (Plastic Leaded Chip Carrier) Packages
- (See Packaging Spec, Order #231369)

The 82C284 is a clock generator/driver which provides clock signals for 80286 processors and support components. It also contains logic to supply READY to the CPU from either asynchronous or synchronous sources and synchronous RESET from an asynchronous input.

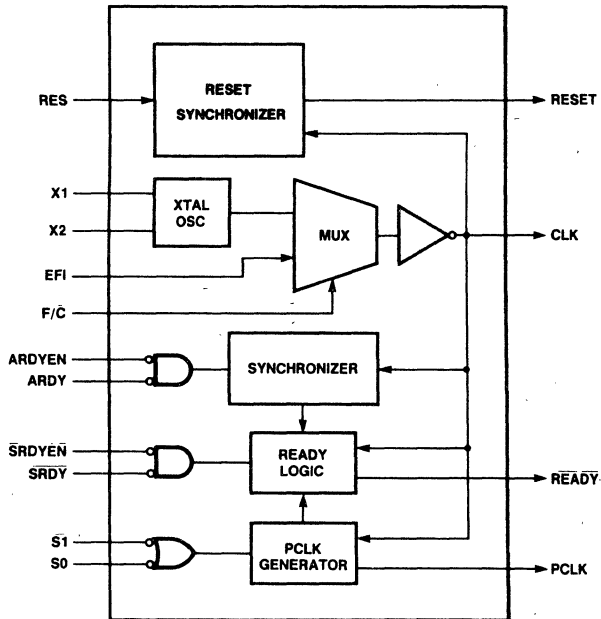
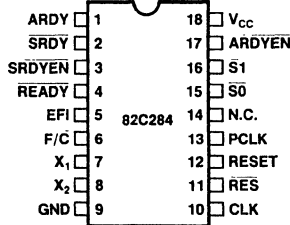


Figure 1. 82C284 Block Diagram

210453-1

18-Lead Cerdip

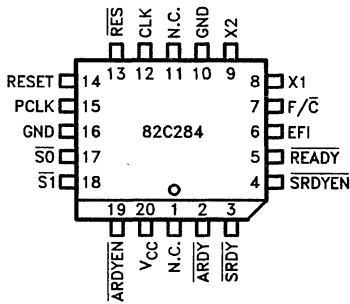


210453-2

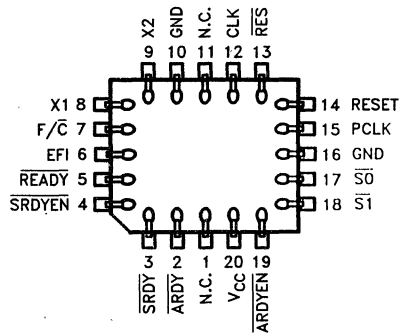
P.C. Board Views—As viewed from the component side of the P.C. Board.

Component Pad Views—As viewed from under-side of component when mounted on the board.

20 Pin PLCC



210453-18



210453-19

NOTE:

1. N.C. Signals must not be connected.

Figure 2. 82C284 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for the 82C284 clock generator.

Symbol	Type	Name and Function
CLK	O	SYSTEM CLOCK is the signal used by the processor and support devices which must be synchronous with the processor. The frequency of the CLK output has twice the desired internal processor clock frequency. CLK can drive both TTL and MOS level inputs.
F/ \bar{C}	I	FREQUENCY/CRYSTAL SELECT is a strapping option to select the source for the CLK output. When F/ \bar{C} is strapped LOW, the internal crystal oscillator drives CLK. When F/ \bar{C} is strapped HIGH, the EFI input drives the CLK output.
X1, X2	I	CRYSTAL IN are the pins to which a parallel resonant fundamental mode crystal is attached for the internal oscillator. When F/ \bar{C} is LOW, the internal oscillator will drive the CLK output at the crystal frequency. The crystal frequency must be twice the desired internal processor clock frequency.
EFI	I	EXTERNAL FREQUENCY IN drives CLK when the F/ \bar{C} input is strapped HIGH. The EFI input frequency must be twice the desired internal processor clock frequency.
PCLK	O	PERIPHERAL CLOCK is an output which provides a 50% duty cycle clock with 1/2 the frequency of CLK. PCLK will be in phase with the internal processor clock following the first bus cycle after the processor has been reset.
$\bar{A}RDYEN$	I	ASYNCHRONOUS READY ENABLE is an active LOW input which qualifies the $\bar{A}RDY$ input. $\bar{A}RDYEN$ selects $\bar{A}RDY$ as the source of ready for the current bus cycle. Inputs to $\bar{A}RDYEN$ may be applied asynchronously to CLK. Setup and hold times are given to assure a guaranteed response to synchronous inputs.
$\bar{A}RDY$	I	ASYNCHRONOUS READY is an active LOW input used to terminate the current bus cycle. The $\bar{A}RDY$ input is qualified by $\bar{A}RDYEN$. Inputs to $\bar{A}RDY$ may be applied asynchronously to CLK. Setup and hold times are given to assure a guaranteed response to synchronous outputs.
$\bar{S}RDYEN$	I	SYNCHRONOUS READY ENABLE is an active LOW input which qualifies $\bar{S}RDY$. $\bar{S}RDYEN$ selects $\bar{S}RDY$ as the source for $\bar{R}EADY$ to the CPU for the current bus cycle. Setup and hold times must be satisfied for proper operation.
$\bar{S}RDY$	I	SYNCHRONOUS READY is an active LOW input used to terminate the current bus cycle. The $\bar{S}RDY$ input is qualified by the $\bar{S}RDYEN$ input. Setup and hold times must be satisfied for proper operation.
$\bar{R}EADY$	O	READY is an active LOW output which signals the current bus cycle is to be completed. The $\bar{S}RDY$, $\bar{S}RDYEN$, $\bar{A}RDY$, $\bar{A}RDYEN$, $\bar{S}1$, $\bar{S}0$ and $\bar{R}ES$ inputs control $\bar{R}EADY$ as explained later in the $\bar{R}EADY$ generator section. $\bar{R}EADY$ is an open drain output requiring an external pull-up resistor.

Table 1. Pin Description (Continued)

The following pin function descriptions are for the 82C284 clock generator.

Symbol	Type	Name and Function
$\overline{S0}, \overline{S1}$	I	STATUS input prepare the 82C284 for a subsequent bus cycle. $\overline{S0}$ and $\overline{S1}$ synchronize PCLK to the internal processor clock and control READY . These inputs have internal pull-up resistors to keep them HIGH if nothing is driving them. Setup and hold times must be satisfied for proper operation.
RESET	O	RESET is an active HIGH output which is derived from the \overline{RES} input. RESET is used to force the system into an initial state. When RESET is active, READY will be active (LOW).
\overline{RES}	I	RESET IN is an active LOW input which generates the system reset signal, RESET . Signals to \overline{RES} may be applied asynchronously to CLK. Setup and hold times are given to assure a guaranteed response to synchronous inputs.
V _{CC}		SYSTEM POWER: +5V Power Supply
GND		SYSTEM GROUND: 0V

FUNCTIONAL DESCRIPTION

Introduction

The 82C284 generates the clock, ready, and reset signals required for 80286 processors and support components. The 82C284 contains a crystal controlled oscillator, clock generator, peripheral clock generator, Multibus ready synchronization logic and system reset generation logic.

Clock Generator

The CLK output provides the basic timing control for an 80286 system. CLK has output characteristics sufficient to drive MOS devices. CLK is generated by either an internal crystal oscillator or an external source as selected by the F/C strapping option. When F/C is LOW, the crystal oscillator drives the CLK output. When F/C is HIGH, the EFI input drives the CLK output.

The 82C284 provides a second clock output, PCLK, for peripheral devices. PCLK is CLK divided by two. PCLK has a duty cycle of 50% and MOS output drive characteristics. PCLK is normally synchronized to the internal processor clock.

After reset, the PCLK signal may be out of phase with the internal processor clock. The $\overline{S1}$ and $\overline{S0}$ signals of the first bus cycle are used to synchronize

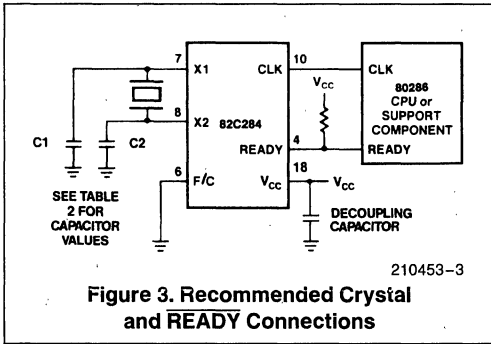
PCLK to the internal processor clock. The phase of the PCLK output changes by extending its HIGH time beyond one system clock (see waveforms). PCLK is forced HIGH whenever either $\overline{S0}$ or $\overline{S1}$ were active (LOW) for the two previous CLK cycles. PCLK continues to oscillate when both $\overline{S0}$ and $\overline{S1}$ are HIGH.

Since the phase of the internal processor clock will not change except during reset, the phase of PCLK will not change except during the first bus cycle after reset.

Oscillator

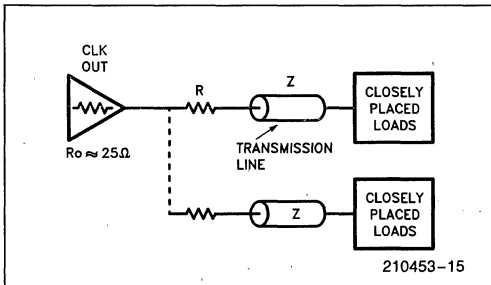
The oscillator circuit of the 82C284 is a linear Pierce oscillator which requires an external parallel resonant, fundamental mode, crystal. The output of the oscillator is internally buffered. The crystal frequency chosen should be twice the required internal processor clock frequency. The crystal should have a typical load capacitance of 32 pF.

X1 and X2 are the oscillator crystal connections. For stable operation of the oscillator, two loading capacitors are recommended, as shown in Table 2. The sum of the board capacitance and loading capacitance should equal the values shown. It is advisable to limit stray board capacitances (not including the effect of the loading capacitors or crystal capacitance) to less than 10 pF between the X1 and X2 pins. Decouple V_{CC} and GND as close to the 82C284 as possible.



CLK Termination

Due to the CLK output having a very fast rise and fall time, it is recommended to properly terminate the CLK line at frequencies above 10 MHz to avoid signal reflections and ringing. Termination is accomplished by inserting a small resistor (typically 10Ω – 74Ω) in series with the output, as shown in Figure 4. This is known as series termination. The resistor value plus the circuit output impedance should be made equal to the impedance of the transmission line.

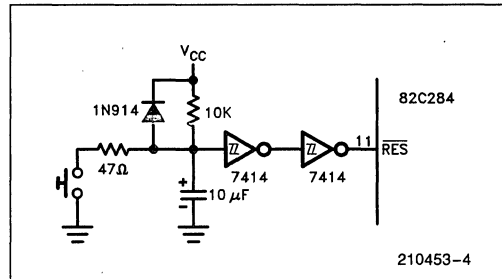


Reset Operation

The reset logic provides the RESET output to force the system into a known, initial state. When the RES input is active (LOW), the RESET output becomes active (HIGH). $\overline{\text{RES}}$ is synchronized internally at the falling edge of CLK before generating the RESET output (see waveforms). Synchronization of the RES input introduces a one or two CLK delay before affecting the RESET output.

At power up, a system does not have a stable V_{CC} and CLK. To prevent spurious activity, $\overline{\text{RES}}$ should

be asserted until V_{CC} and CLK stabilize at their operating values. 80286 processors and support components also require their RESET inputs be HIGH a minimum of 16 CLK cycles. A network such as shown in Figure 5 will keep RES LOW long enough to satisfy both needs.



Ready Operation

The 82C284 accepts two ready sources for the system ready signal which terminates the current bus cycle. Either a synchronous ($\overline{\text{SRDY}}$) or asynchronous ready ($\overline{\text{ARDY}}$) source may be used. Each ready input has an enable ($\overline{\text{SRDYEN}}$ and $\overline{\text{ARDYEN}}$) for selecting the type of ready source required to terminate the current bus cycle. An address decoder would normally select one of the enable inputs.

$\overline{\text{READY}}$ is enabled (LOW), if either $\overline{\text{SRDY}} + \overline{\text{SRDYEN}} = 0$ or $\overline{\text{ARDY}} + \overline{\text{ARDYEN}} = 0$ when sampled by the 82C284 $\overline{\text{READY}}$ generation logic. $\overline{\text{READY}}$ will remain active for at least two CLK cycles.

The $\overline{\text{READY}}$ output has an open-drain driver allowing other ready circuits to be wire or'ed with it, as shown in Figure 3. The $\overline{\text{READY}}$ signal of an 80286 system requires an external pull-up resistor. To force the $\overline{\text{READY}}$ signal inactive (HIGH) at the start of a bus cycle, the $\overline{\text{READY}}$ output floats when either $\overline{\text{S1}}$ or $\overline{\text{S0}}$ are sampled LOW at the falling edge of CLK. Two system clock periods are allowed for the pull-up resistor to pull the $\overline{\text{READY}}$ signal to V_{IH} . When RESET is active, $\overline{\text{READY}}$ is forced active one CLK later (see waveforms).

Figure 6 illustrates the operation of $\overline{\text{SRDY}}$ and $\overline{\text{SRDYEN}}$. These inputs are sampled on the falling edge of CLK when $\overline{\text{S1}}$ and $\overline{\text{S0}}$ are inactive and PCLK

is HIGH. $\overline{\text{READY}}$ is forced active when both $\overline{\text{SRDY}}$ and $\overline{\text{SRDYEN}}$ are sampled as LOW.

Figure 7 shows the operation of $\overline{\text{ARDY}}$ and $\overline{\text{ARDYEN}}$. These inputs are sampled by an internal synchronizer at each falling edge of CLK. The output of the synchronizer is then sampled when PCLK is HIGH. If the synchronizer resolved both the $\overline{\text{ARDY}}$

and $\overline{\text{ARDYEN}}$ as active, the $\overline{\text{SRDY}}$ and $\overline{\text{SRDYEN}}$ inputs are ignored. Either $\overline{\text{ARDY}}$ or $\overline{\text{ARDYEN}}$ must be HIGH at the end of T_S (see Figure 7).

$\overline{\text{READY}}$ remains active until either $\overline{\text{S1}}$ or $\overline{\text{S0}}$ are sampled LOW, or the ready inputs are sampled as inactive.

Table 2. 82C284 Crystal Loading Capacitance Values

Crystal Frequency	C1 Capacitance (Pin 7)	C2 Capacitance (Pin 8)
1 to 8 MHz	60 pF	40 pF
8 to 20 MHz	25 pF	15 pF
Above 20 MHz	15 pF	15 pF

NOTE:
Capacitance values must include stray board capacitance.

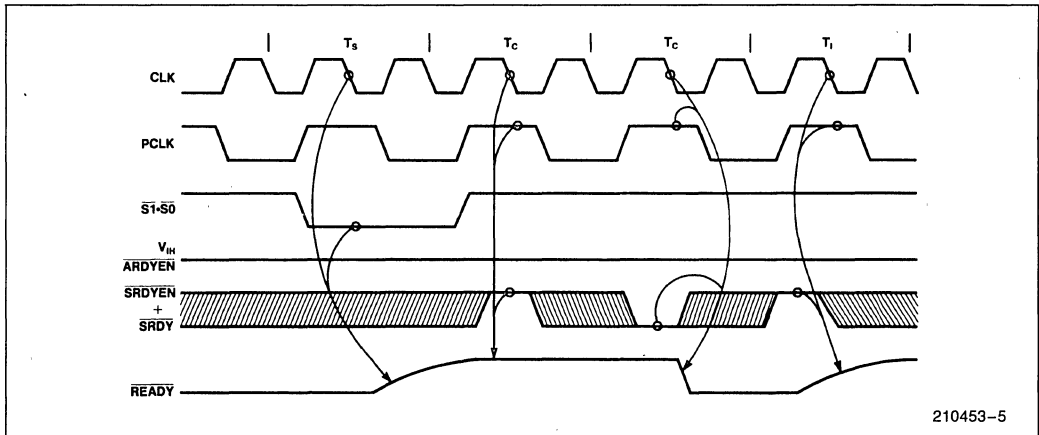


Figure 6. Synchronous Ready Operation

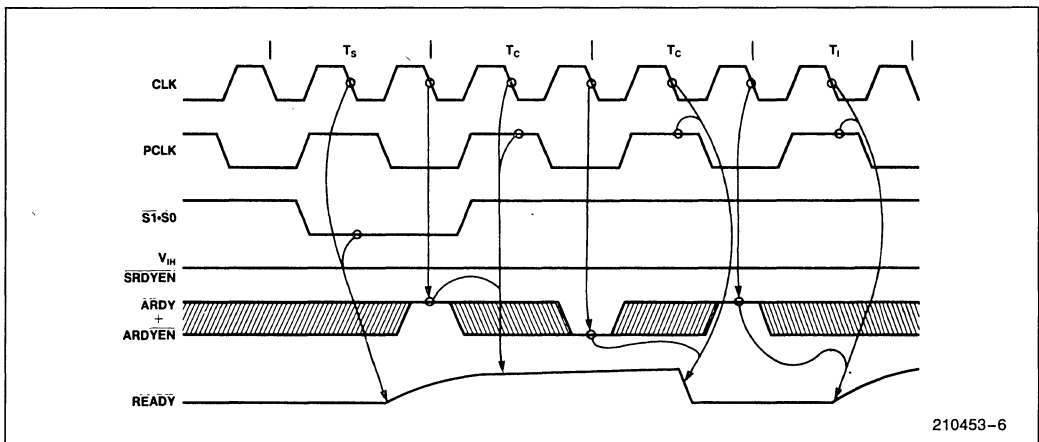


Figure 7. Asynchronous Ready Operation

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias 0°C to +70°C
 Storage Temperature -65°C to +150°C
 All Output and Supply Voltages -0.5V to +7V
 All Input Voltages -1.0V to +5.5V
 Power Dissipation 1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_{CASE} = 0°C to +85°C, * V_{CC} = 5V ±5%

Symbol	Parameter	Min	Max	Unit	Test Condition
V _{IL}	Input LOW Voltage		0.8	V	
V _{IH}	Input HIGH Voltage	2.0		V	
V _{IHR}	$\overline{\text{RES}}$ and EFl Input HIGH Voltage	2.6		V	
V _{OL}	RESET, PCLK Output LOW Voltage		0.45	V	I _{OL} = 5 mA
V _{OH}	RESET, PCLK Output HIGH Voltage	2.4		V	I _{OH} = -1 mA
		V _{CC} -0.5		V	I _{OH} = -0.2 mA
V _{OLR}	$\overline{\text{READY}}$, Output LOW Voltage		0.45	V	I _{OL} = 9 mA
V _{OLC}	CLK Output LOW Voltage		0.45	V	I _{OL} = 5 mA
V _{OHc}	CLK Output HIGH Voltage	4.0		V	I _{OH} = - 800 μA
I _{IL}	Input Sustaining Current on S0 and S1 Pins	- 30	- 500	μA	V _{IN} = 0V
I _{LI}	Input Leakage Current		± 10	μA	0 ≤ V _{IN} ≤ V _{CC} (1)
I _{CC}	Power Supply Current		75	mA	at 25 MHz Output CLK Frequency
C _I	Input Capacitance		10	pF	F _C = 1 MHz

*T_A is guaranteed from 0°C to +70°C as long as T_{CASE} is not exceeded.

NOTE:

- Status lines S0 and S1 excluded because they have internal pull-up resistors.

A.C. CHARACTERISTICS $V_{CC} = 5V \pm 5\%$, $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$.*

Timings are referenced to 0.8V and 2.0V points of signals as illustrated in the datasheet waveforms, unless otherwise noted.

82C284 A.C. Timing Parameters

Symbol	Parameter	8.0 MHz		10.0 MHz		12.5 MHz		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
1	EFI to CLK Delay		25		25		25	ns	At 1.5V (1)
2	EFI LOW Time	28		22.5		13		ns	At 1.5V (1, 7)
3	EFI HIGH Time	28		22.5		22		ns	At 1.5V (1, 7)
4	CLK Period	62	500	50	500	40	500	ns	
5	CLK LOW Time	15		12		11		ns	At 1.0V (1, 2, 7, 8, 9, 10)
6	CLK HIGH Time	25		16		13		ns	At 3.6V (1, 2, 7, 8, 9, 10)
7	CLK Rise Time		10		8		8	ns	1.0V to 3.6V (1, 2, 10, 11)
8	CLK Fall Time		10		8		8	ns	3.6V to 1.0V (1, 9, 10, 11)
9	Status Setup Time	22		—		—		ns	(Note 1)
9a	Status Setup Time for Status Going Active	—		20		22		ns	(Note 1)
9b	Status Setup Time for Status Going Inactive	—		20		18		ns	(Note 1)
10	Status Hold Time	1		1		3		ns	(Note 1)
11	\overline{SRDY} or \overline{SRDYEN} Setup Time	20		18		18		ns	(Note 1)
12	\overline{SRDY} or \overline{SRDYEN} Hold Time	0		2		2		ns	(Notes 1, 11)
13	\overline{ARDY} or \overline{ARDYEN} Setup Time	0		0		0		ns	(Notes 1, 3)
14	\overline{ARDY} or \overline{ARDYEN} Hold Time	30		30		25		ns	(Notes 1, 3)
15	\overline{RES} Setup Time	20		20		18		ns	(Notes 1, 3)
16	\overline{RES} Hold Time	10		10		8		ns	(Notes 1, 3)
17	\overline{READY} Inactive Delay	5		5		5		ns	At 0.8V (4)
18	\overline{READY} Active Delay	0	24	0	24	0	18	ns	At 0.8V (4)
19	PCLK Delay	0	45	0	35	0	23	ns	(Note 5)
20	RESET Delay	5	34	5	27	3	22	ns	(Note 5)
21	PCLK LOW Time	t4-20		t4-20		T4-20		ns	(Notes 5, 6)
22	PCLK HIGH Time	t4-20		t4-20		T4-20		ns	(Notes 5, 6)

 * T_A is guaranteed from $0^{\circ}C$ to $70^{\circ}C$ as long as T_{CASE} is not exceeded.

NOTES:

- CLK loading: $C_L = 100$ pF. The 82C284's X1 and X2 inputs are designed primarily for parallel-resonant crystals. Serial-resonant crystals may also be used, however, they may oscillate up to 0.01% faster than their nominal frequencies when used with the 82C284. For either type of crystal, capacitive loading should be as specified by Table 2.
- With the internal crystal oscillator using recommended crystal and capacitive loading; or with the EFI input meeting specifications t2 and t3. The recommended crystal loading for CLK frequencies of 8 MHz–20 MHz are 25 pF from pin X₁ to ground, and 15 pF from pin X₂ to ground; for CLK frequencies above 20 MHz 15 pF from pin X₁ to ground, and 15 pF from pin X₂ to ground. These recommended values are ± 5 pF and include all stray capacitance. Decouple V_{CC} and GND as close to the 82C284 as possible.
- This is an asynchronous input. This specification is given for testing purposes only, to assure recognition at specific CLK edge.

NOTES:

4. Pull-up Resistor values for READY Pin:

CPU Frequency	8 MHz	10 MHz	12.5 MHz
Resistor	910Ω	700Ω	600Ω
CL	150 pF	150 pF	150 pF
I _{OL}	7 mA	7 mA	9 mA

5. PCLK and RESET loading: C_L = 75 pF.

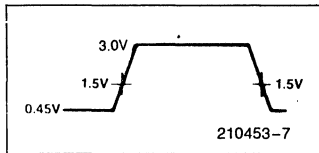
6. t₄ refers to any allowable CLK period.

7. When driving the 82C284 with EFI, provide minimum EFI HIGH and LOW times as follows:

CLK Output Frequency	16 MHz	20 MHz	25 MHz
Min. Required EFI HIGH Time	28 ns	22.5 ns	22 ns
Min. Required EFI LOW Time	28 ns	22.5 ns	13 ns

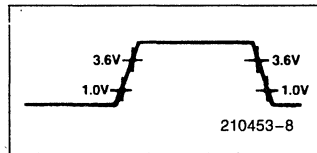
8. When using a crystal (with recommended capacitive loading per Table 2) appropriate for the speed of the 80286, CLK output HIGH and LOW times guaranteed to meet the 80286 requirements.

Reset Drive EFI Drive and Measurement Points



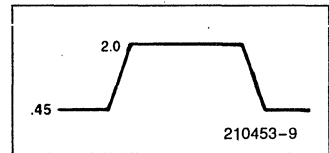
Note 9

CLK Output Measurement Points

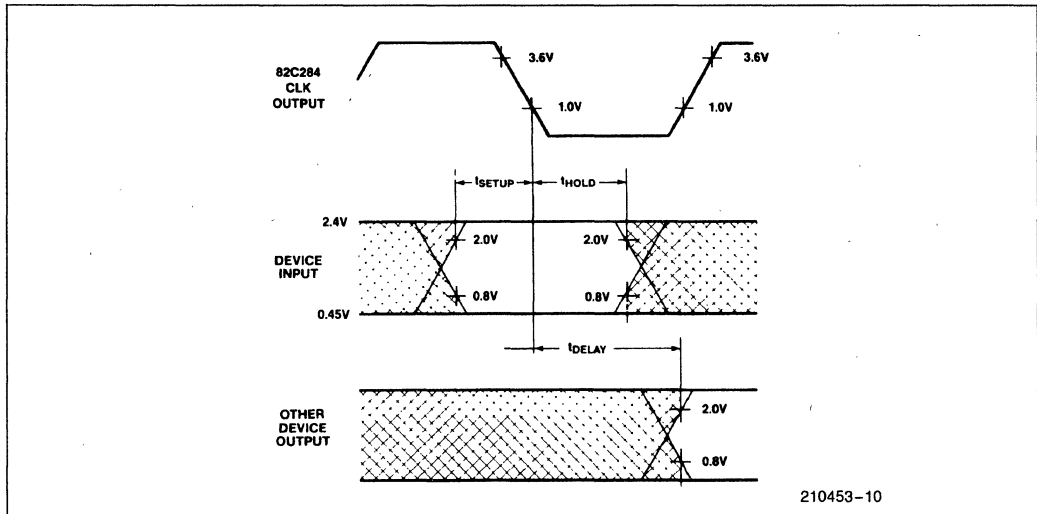


Note 10

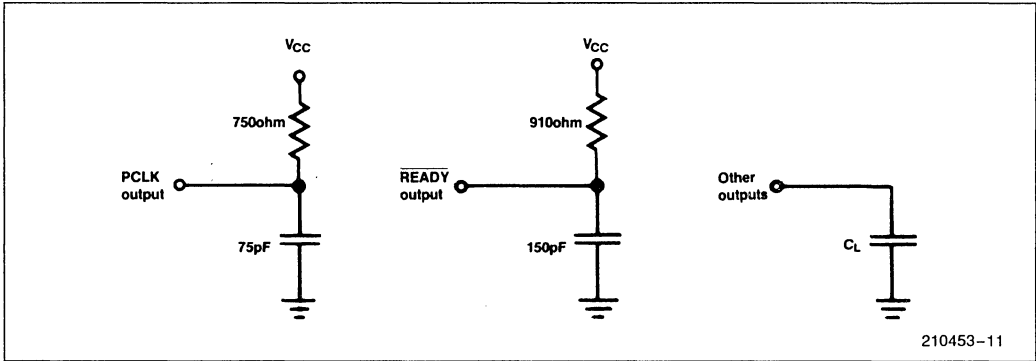
F/C Drive Points



Note 11



Note 12. AC Setup, Hold and Delay Time Measurement—General

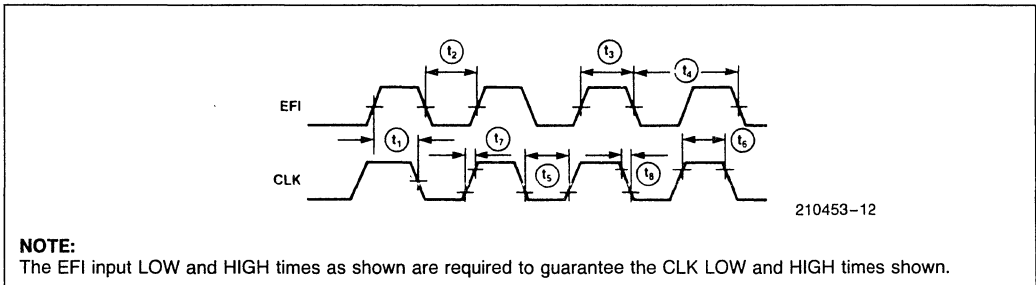


210453-11

Note 13. AC Test Loading on Outputs

WAVEFORMS

CLK as a Function of EFI

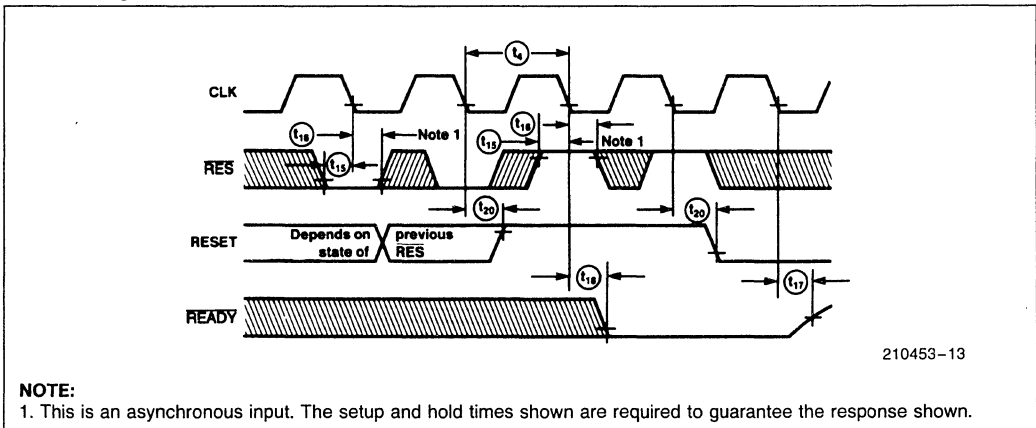


210453-12

NOTE:

The EFI input LOW and HIGH times as shown are required to guarantee the CLK LOW and HIGH times shown.

RESET and READY Timing as a Function of RES with S1, S0, ARDY + ARDYEN, and SRDY + SRDYEN High



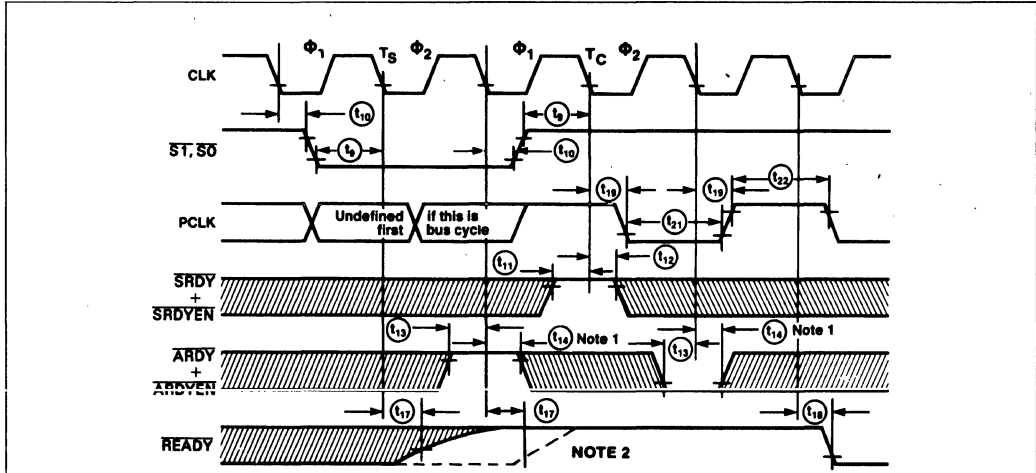
210453-13

NOTE:

1. This is an asynchronous input. The setup and hold times shown are required to guarantee the response shown.

WAVEFORMS (Continued)

READY and PCLK Timing with RES High

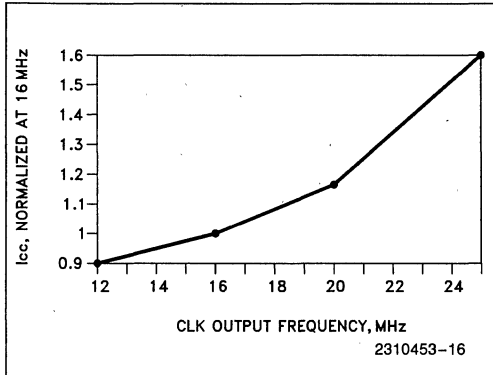


210453-14

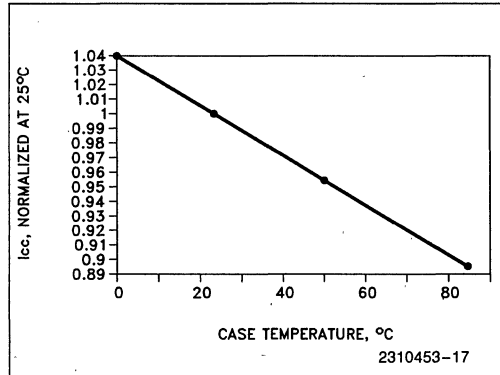
NOTES:

1. This is an asynchronous input. The setup and hold times shown are required to guarantee the response shown.
2. If SRDY + SRDYEN or ARDY + ARDYEN are active before and/or during the first bus cycle after RESET, READY may not be deasserted until after the falling edge of ϕ_2 of T_S .

I_{CC} vs Frequency @ Nominal Conditions



I_{CC} vs Case Temperature @ 25 MHz



DATA SHEET REVISION REVIEW

The following list represents key differences between this and the -009 data sheet. Please review this summary carefully.

1. The AC timing parameter $\overline{\text{SRDY}}$ or $\overline{\text{SRDYEN}}$ setup time (T11) has been changed to 18 ns for the 10 and 12.5 MHz parts and 20 ns for the 8 MHz part.



i486™ MICROPROCESSOR

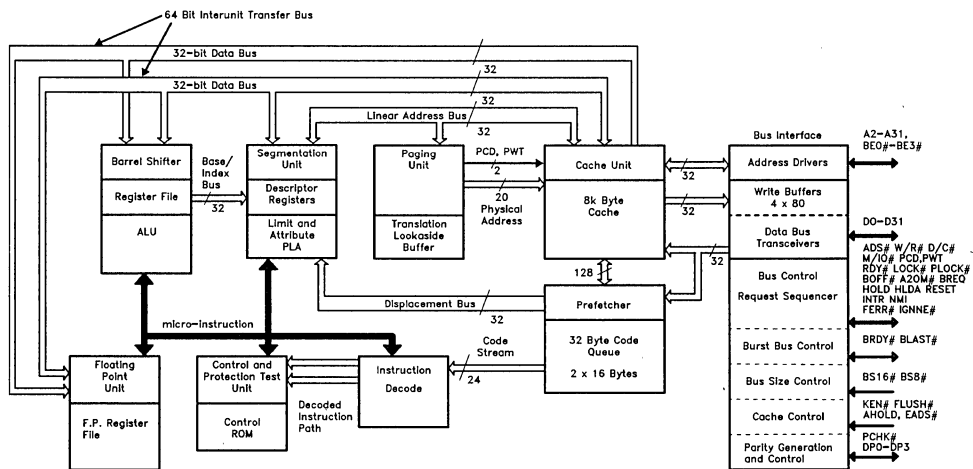
- **Binary Compatible with Large Software Base**
 - MS-DOS*, OS/2**, Windows
 - UNIX*** System V/386
 - iRMX®, iRMK™ Kernels
- **High Integration Enables On-Chip**
 - 8 Kbyte Code and Data Cache
 - Floating Point Unit
 - Paged, Virtual Memory Management
- **Easy To Use**
 - Built-In Self Test
 - Hardware Debugging Support
 - Intel Software Support
 - Extensive Third Party Software Support
- **High Performance Design**
 - Frequent Instructions Execute in One Clock
 - 25 MHz and 33 MHz Clock Frequencies
 - 106 Mbyte/Sec Burst Bus
 - CHMOS IV Process Technology
- **Complete 32-Bit Architecture**
 - Address and Data Buses
 - Registers
- **Multiprocessor Support**
 - Multiprocessor Instructions
 - Cache Consistency Protocols
 - Support for Second Level Cache

The i486™ CPU offers the highest performance for DOS, OS/2, Windows and UNIX System V/386 applications. It is 100% binary compatible with the 386™ CPU. One million transistors integrate cache memory, floating point hardware and memory management on-chip while retaining binary compatibility with previous members of the 86 architectural family. Frequently used instructions execute in one cycle resulting in RISC performance levels. An 8 Kbyte unified code and data cache combined with a 106 Mbyte/Sec burst bus at 33.3 MHz ensure high system throughput even with inexpensive DRAMs.

New features enhance multiprocessing systems. New instructions speed manipulation of memory based semaphores. On-chip hardware ensures cache consistency and provides hooks for multilevel caches.

The built in self test extensively tests on-chip logic, cache memory and the on-chip paging translation cache. Debug features include breakpoint traps on code execution and data accesses.

i486™ Microprocessor Pipelined 32-Bit Microarchitecture



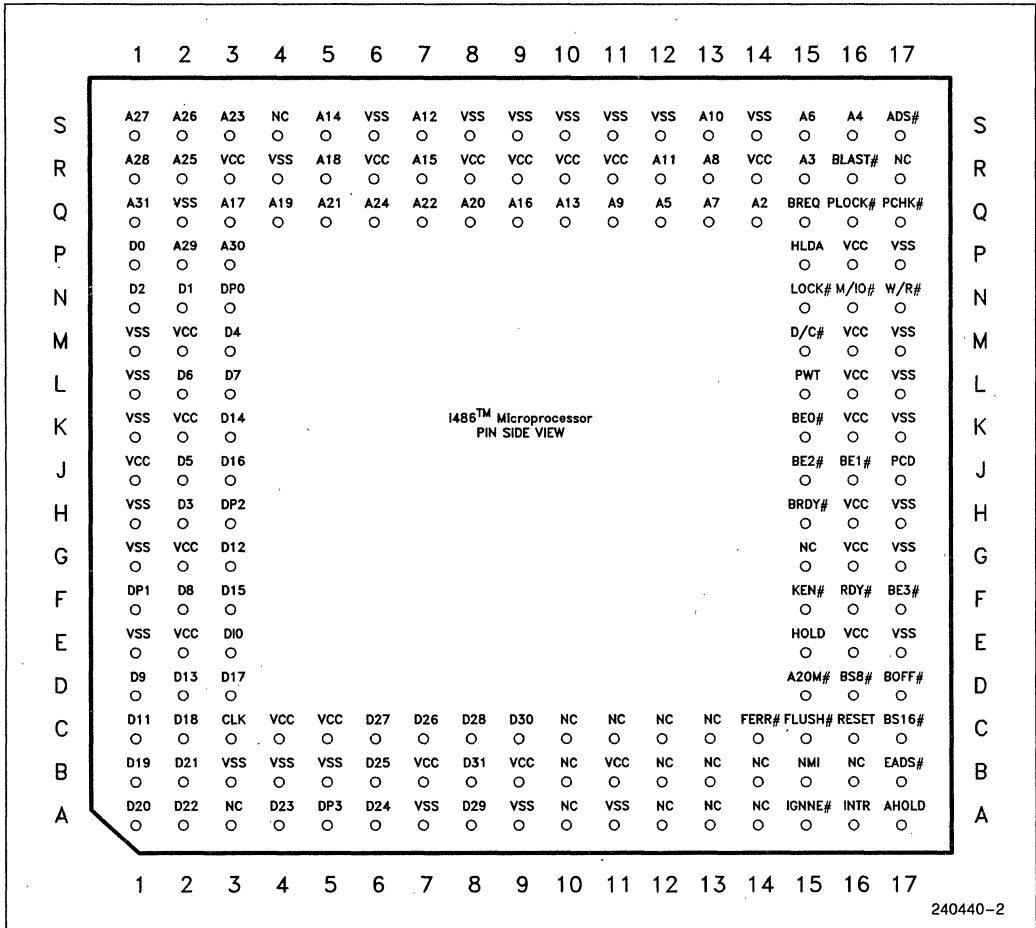
240440-1

iRMX, iRMK, 386, 387, 486, i486 are trademarks of Intel Corporation.

*MS-DOS® is a registered trademark of Microsoft Corporation.

**OS/2™ is a trademark of Microsoft Corporation.

***UNIX™ is a trademark of AT&T.



240440-2

Figure 1.1

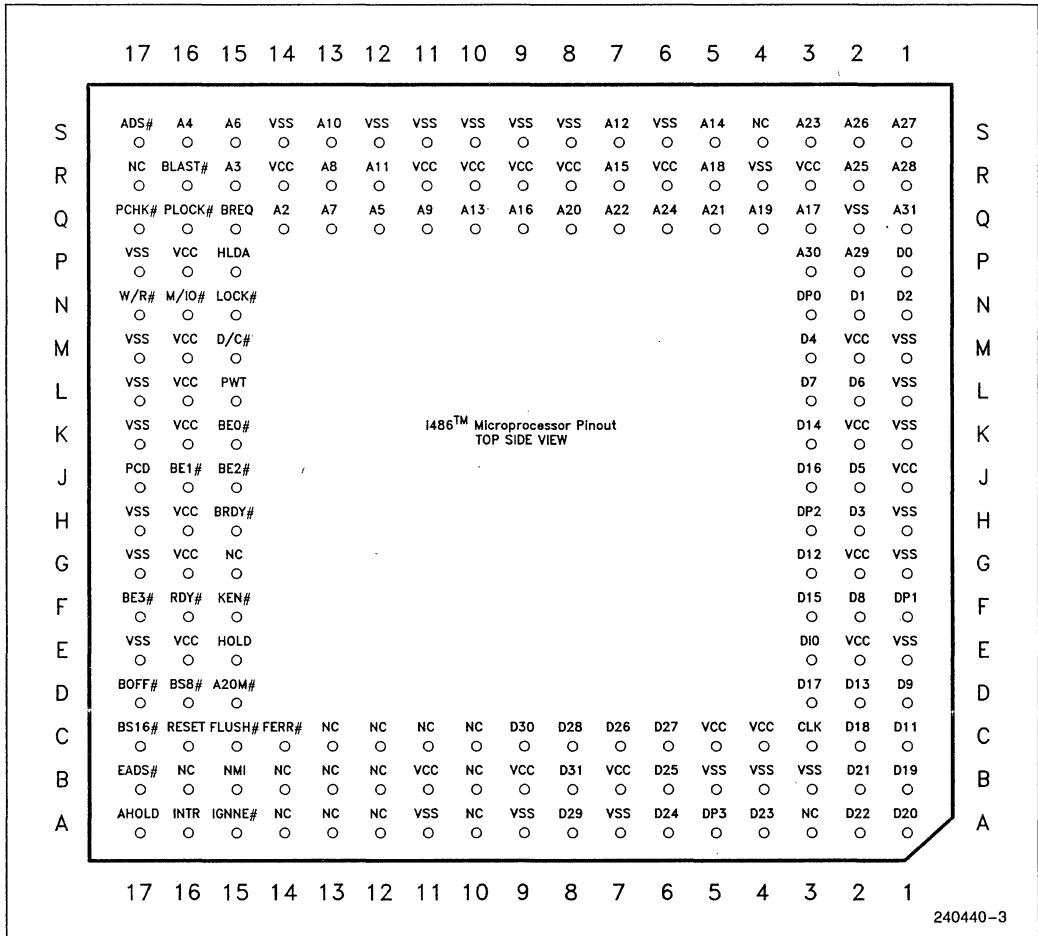


Figure 1.2

Pin Cross Reference by Pin Name

Address		Data		Control		N/C	V _{CC}	V _{SS}
A ₂	Q14	D ₀	P1	A20M#	D15	A3	B7	A7
A ₃	R15	D ₁	N2	ADS#	S17	A10	B9	A9
A ₄	S16	D ₂	N1	AHOLD	A17	A12	B11	A11
A ₅	Q12	D ₃	H2	BE0#	K15	A13	C4	B3
A ₆	S15	D ₄	M3	BE1#	J16	A14	C5	B4
A ₇	Q13	D ₅	J2	BE2#	J15	B10	E2	B5
A ₈	R13	D ₆	L2	BE3#	F17	B12	E16	E1
A ₉	Q11	D ₇	L3	BLAST#	R16	B13	G2	E17
A ₁₀	S13	D ₈	F2	BOFF#	D17	B14	G16	G1
A ₁₁	R12	D ₉	D1	BRDY#	H15	B16	H16	G17
A ₁₂	S7	D ₁₀	E3	BREQ#	Q15	C10	J1	H1
A ₁₃	Q10	D ₁₁	C1	BS8#	D16	C11	K2	H17
A ₁₄	S5	D ₁₂	G3	BS16#	C17	C12	K16	K1
A ₁₅	R7	D ₁₃	D2	CLK	C3	C13	L16	K17
A ₁₆	Q9	D ₁₄	K3	D/C#	M15	G15	M2	L1
A ₁₇	Q3	D ₁₅	F3	DP0	N3	R17	M16	L17
A ₁₈	R5	D ₁₆	J3	DP1	F1	S4	P16	M1
A ₁₉	Q4	D ₁₇	D3	DP2	H3		R3	M17
A ₂₀	Q8	D ₁₈	C2	DP3	A5		R6	P17
A ₂₁	Q5	D ₁₉	B1	EADS#	B17		R8	Q2
A ₂₂	Q7	D ₂₀	A1	FERR#	C14		R9	R4
A ₂₃	S3	D ₂₁	B2	FLUSH#	C15		R10	S6
A ₂₄	Q6	D ₂₂	A2	HLDA	P15		R11	S8
A ₂₅	R2	D ₂₃	A4	HOLD	E15		R14	S9
A ₂₆	S2	D ₂₄	A6	IGNNE#	A15			S10
A ₂₇	S1	D ₂₅	B6	INTR	A16			S11
A ₂₈	R1	D ₂₆	C7	KEN#	F15			S12
A ₂₉	P2	D ₂₇	C6	LOCK#	N15			S14
A ₃₀	P3	D ₂₈	C8	M/IO#	N16			
A ₃₁	Q1	D ₂₉	A8	NMI	B15			
		D ₃₀	C9	PCD	J17			
		D ₃₁	B8	PCHK#	Q17			
				PWT	L15			
				PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				W/R#	N17			

QUICK PIN REFERENCE

What follows is a brief pin description. For detailed signal descriptions refer to Section 6.

Symbol	Type	Name and Function																																				
CLK	I	<i>Clock</i> provides the fundamental timing and the internal operating frequency for the 486 microprocessor. All external timing parameters are specified with respect to the rising edge of CLK.																																				
ADDRESS BUS																																						
A31–A4 A2–A3	I/O O	A31–A2 are the <i>address lines</i> of the microprocessor. A31–A2, together with the byte enables BE0#–BE3#, define the physical area of memory or input/output space accessed. Address lines A31–A4 are used to drive addresses into the microprocessor to perform cache line invalidations. Input signals must meet setup and hold times t_{22} and t_{23} . A31–A2 are not driven during bus or address hold.																																				
BE0–3#	O	The <i>byte enable</i> signals indicate active bytes during read and write cycles. During the first cycle of a cache fill, the external system should assume that all byte enables are active. BE3# applies to D24–D31, BE2# applies to D16–D23, BE1# applies to D8–D15 and BE0# applies to D0–D7. BE0#–BE3# are active LOW and are not driven during bus hold.																																				
DATA BUS																																						
D31–D0	I/O	These are the <i>data lines</i> for the 486 microprocessor. Lines D0–D7 define the least significant byte of the data bus while lines D24–D31 define the most significant byte of the data bus. These signals must meet setup and hold times t_{22} and t_{23} for proper operation on reads. These pins are driven during the second and subsequent clocks of write cycles.																																				
DATA PARITY																																						
DP0–DP3	I/O	There is one <i>data parity</i> pin for each byte of the data bus. Data parity is generated on all write data cycles with the same timing as the data driven by the 486 microprocessor. Even parity information must be driven back into the microprocessor on the data parity pins with the same timing as read information to insure that the correct parity check status is indicated by the 486 microprocessor. The signals read on these pins do not affect program execution. Input signals must meet setup and hold times t_{22} and t_{23} . DP0–DP3 should be connected to V_{CC} through a pullup resistor in systems which do not use parity. DP0–DP3 are active HIGH and are driven during the second and subsequent clocks of write cycles.																																				
PCHK#	O	<i>Parity Status</i> is driven on the PCHK# pin the clock after ready for read operations. The parity status is for data sampled at the end of the previous clock. A parity error is indicated by PCHK# being LOW. Parity status is only checked for enabled bytes as indicated by the byte enable and bus size signals. PCHK# is valid only in the clock immediately after read data is returned to the microprocessor. At all other times PCHK# is inactive (HIGH). PCHK# is never floated.																																				
BUS CYCLE DEFINITION																																						
M/IO# D/C# W/R#	O O O	The <i>memory/input-output, data/control</i> and <i>write/read</i> lines are the primary bus definition signals. These signals are driven valid as the ADS# signal is asserted.																																				
		<table border="1"> <thead> <tr> <th>M/IO#</th> <th>D/C#</th> <th>W/R#</th> <th>Bus Cycle Initiated</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>Interrupt Acknowledge</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>Halt/Special Cycle</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>I/O Read</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>I/O Write</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>Code Read</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>Reserved</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>Memory Read</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>Memory Write</td> </tr> </tbody> </table>	M/IO#	D/C#	W/R#	Bus Cycle Initiated	0	0	0	Interrupt Acknowledge	0	0	1	Halt/Special Cycle	0	1	0	I/O Read	0	1	1	I/O Write	1	0	0	Code Read	1	0	1	Reserved	1	1	0	Memory Read	1	1	1	Memory Write
M/IO#	D/C#	W/R#	Bus Cycle Initiated																																			
0	0	0	Interrupt Acknowledge																																			
0	0	1	Halt/Special Cycle																																			
0	1	0	I/O Read																																			
0	1	1	I/O Write																																			
1	0	0	Code Read																																			
1	0	1	Reserved																																			
1	1	0	Memory Read																																			
1	1	1	Memory Write																																			
		The bus definition signals are not driven during bus hold and follow the timing of the address bus. Refer to Section 7.2.11 for a description of the special bus cycles.																																				

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
BUS CYCLE DEFINITION (Continued)		
LOCK #	O	The <i>bus lock</i> pin indicates that the current bus cycle is locked. The 486 microprocessor will not allow a bus hold when LOCK # is asserted (but address holds are allowed). LOCK # goes active in the first clock of the first locked bus cycle and goes inactive after the last clock of the last locked bus cycle. The last locked cycle ends when ready is returned. LOCK # is active LOW and is not driven during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN # is returned active.
PLOCK #	O	The <i>pseudo-lock</i> pin indicates that the current bus transaction requires more than one bus cycle to complete. Examples of such operations are floating point long reads and writes (64 bits), segment table descriptor reads (64 bits), in addition to cache line fills (128 bits). The 486 microprocessor will drive PLOCK # active until the addresses for the last bus cycle of the transaction have been driven regardless of whether RDY # or BRDY # have been returned. Normally PLOCK # and BLAST # are inverse of each other. However during the first bus cycle of a 64-bit floating point write, both PLOCK # and BLAST # will be asserted. PLOCK # is a function of the BS8 #, BS16 # and KEN # inputs. PLOCK # should be sampled only in the clock ready is returned. PLOCK # is active LOW and is not driven during bus hold.
BUS CONTROL		
ADS #	O	The <i>address status</i> output indicates that a valid bus cycle definition and address are available on the cycle definition lines and address bus. ADS # is driven active in the same clock as the addresses are driven. ADS # is active LOW and is not driven during bus hold.
RDY #	I	The <i>non-burst ready</i> input indicates that the current bus cycle is complete. RDY # indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted data from the 486 microprocessor in response to a write. RDY # is ignored when the bus is idle and at the end of the first clock of the bus cycle. RDY # is active during address hold. Data can be returned to the processor while AHOLD is active. RDY # is active LOW, and is not provided with an internal pullup resistor. RDY # must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.
BURST CONTROL		
BRDY #	I	The <i>burst ready input</i> performs the same function during a burst cycle that RDY # performs during a non-burst cycle. BRDY # indicates that the external system has presented valid data in response to a read or that the external system has accepted data in response to a write. BRDY # is ignored when the bus is idle and at the end of the first clock in a bus cycle. BRDY # is sampled in the second and subsequent clocks of a burst cycle. The data presented on the data bus will be strobed into the microprocessor when BRDY # is sampled active. If RDY # is returned simultaneously with BRDY #, BRDY # is ignored and the burst cycle is prematurely aborted. BRDY # is active LOW and is provided with a small pullup resistor. BRDY # must satisfy the setup and hold times t_{16} and t_{17} .
BLAST #	O	The <i>burst last</i> signal indicates that the next time BRDY # is returned the burst bus cycle is complete. BLAST # is active for both burst and non-burst bus cycles. BLAST # is active LOW and is not driven during bus hold.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
INTERRUPTS		
RESET	I	The <i>reset</i> input forces the 486 microprocessor to begin execution at a known state. The microprocessor cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK have reached their proper DC and AC specifications. The RESET pin should remain active during this time to insure proper microprocessor operation. RESET is active HIGH. RESET is asynchronous but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
INTR	I	The <i>maskable interrupt</i> indicates that an external interrupt has been generated. If the internal interrupt flag is set in EFLAGS, active interrupt processing will be initiated. The 486 microprocessor will generate two locked interrupt acknowledge bus cycles in response to the INTR pin going active. INTR must remain active until the interrupt acknowledges have been performed to assure that the interrupt is recognized. INTR is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
NMI	I	The <i>non-maskable interrupt</i> request signal indicates that an external non-maskable interrupt has been generated. NMI is rising edge sensitive. NMI must be held LOW for at least four CLK periods before this rising edge. NMI is not provided with an internal pulldown resistor. NMI is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
BUS ARBITRATION		
BREQ	O	The <i>internal cycle pending</i> signal indicates that the 486 microprocessor has internally generated a bus request. BREQ is generated whether or not the 486 microprocessor is driving the bus. BREQ is active HIGH and is never floated.
HOLD	I	The <i>bus hold request</i> allows another bus master complete control of the 486 microprocessor bus. In response to HOLD going active the 486 microprocessor will float most of its output and input/output pins. HLDA will be asserted after completing the current bus cycle, burst cycle or sequence of locked cycles. The 486 microprocessor will remain in this state until HOLD is deasserted. HOLD is active high and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper operation.
HLDA	O	<i>Hold acknowledge</i> goes active in response to a hold request presented on the HOLD pin. HLDA indicates that the 486 microprocessor has given the bus to another local bus master. HLDA is driven active in the same clock that the 486 microprocessor floats its bus. HLDA is driven inactive when leaving bus hold. HLDA is active HIGH and remains driven during bus hold.
BOFF #	I	The <i>backoff</i> input forces the 486 microprocessor to float its bus in the next clock. The microprocessor will float all pins normally floated during bus hold but HLDA will not be asserted in response to BOFF #. BOFF # has higher priority than RDY # or BRDY #; if both are returned in the same clock, BOFF # takes effect. The microprocessor remains in bus hold until BOFF # is negated. If a bus cycle was in progress when BOFF # was asserted the cycle will be restarted. BOFF # is active LOW and must meet setup and hold times t_{18} and t_{19} for proper operation.
CACHE INVALIDATION		
AHOLD	I	The <i>address hold</i> request allows another bus master access to the 486 microprocessor's address bus for a cache invalidation cycle. The 486 microprocessor will stop driving its address bus in the clock following AHOLD going active. Only the address bus will be floated during address hold, the remainder of the bus will remain active. AHOLD is active HIGH and is provided with a small internal pulldown resistor. For proper operation AHOLD must meet setup and hold times t_{18} and t_{19} .



QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
CACHE INVALIDATION (Continued)		
EADS#	I	This signal indicates that a <i>valid external address</i> has been driven onto the 486 microprocessor address pins. This address will be used to perform an internal cache invalidation cycle. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy setup and hold times t_{12} and t_{13} for proper operation.
CACHE CONTROL		
KEN#	I	The <i>cache enable</i> pin is used to determine whether the current cycle is cacheable. When the 486 microprocessor generates a cycle that can be cached and KEN# is active, the cycle will become a cache line fill cycle. Returning KEN# active one clock before ready during the last read in the cache line fill will cause the line to be placed in the on-chip cache. KEN# is active LOW and is provided with a small internal pullup resistor. KEN# must satisfy setup and hold times t_{14} and t_{15} for proper operation.
FLUSH#	I	The <i>cache flush</i> input forces the 486 microprocessor to flush its entire internal cache. FLUSH# is active low and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition in any specific clock. FLUSH# being sampled low in the clock before the falling edge of RESET causes the 486 microprocessor to enter the tri-state test mode.
PAGE CACHEABILITY		
PWT PCD	O O	The <i>page write-through</i> and <i>page cache disable</i> pins reflect the state of the page attribute bits, PWT and PCD, in the page table entry or page directory entry. If paging is disabled or for cycles that are not paged, PWT and PCD reflect the state of the PWT and PCD bits in control register 3. PWT and PCD have the same timing as the cycle definition pins (M/IO#, D/C# and W/R#). PWT and PCD are active HIGH and are not driven during bus hold. PCD is masked by the cache disable bit (CD) in Control Register 0.
NUMERIC ERROR REPORTING		
FERR#	O	The <i>floating point error</i> pin is driven active when a floating point error occurs. FERR# is similar to the ERROR# pin on the 387™ math coprocessor. FERR# is included for compatibility with systems using DOS type floating point error reporting. FERR# is active LOW, and is not floated during bus hold.
IGNNE#	I	When the <i>ignore numeric error</i> pin is asserted the 486 microprocessor will ignore a numeric error and continue executing non-control floating point instructions. When IGNNE# is deasserted the 486 microprocessor will freeze on a non-control floating point instruction, if a previous floating point instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set. IGNNE# is active LOW and is provided with a small internal pullup resistor. IGNNE# is asynchronous but setup and hold times t_{20} and t_{21} must be met to insure recognition on any specific clock.
BUS SIZE CONTROL		
BS16# BS8#	I I	The <i>bus size 16</i> and <i>bus size 8</i> pins (bus sizing pins) cause the 486 microprocessor to run multiple bus cycles to complete a request from devices that cannot provide or accept 32 bits of data in a single cycle. The bus sizing pins are sampled every clock. The state of these pins in the clock before ready is used by the 486 microprocessor to determine the bus size. These signals are active LOW and are provided with internal pullup resistors. These inputs must satisfy setup and hold times t_{14} and t_{15} for proper operation.
ADDRESS MASK		
A20M#	I	When the <i>address bit 20 mask</i> pin is asserted, the 486 microprocessor masks physical address bit 20 (A20) before performing a lookup to the internal cache or driving a memory cycle on the bus. A20M# emulates the address wraparound at one Mbyte which occurs on the 8086. A20M# is active LOW and should be asserted only when the processor is in real mode. This pin is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For proper operation, A20M# should be sampled high at the falling edge of RESET.

Table 1.1. Output Pins

Name	Active Level	When Floated
BREQ	HIGH	
HLDA	HIGH	
BE0# -BE3#	LOW	Bus Hold
PWT, PCD	HIGH	Bus Hold
W/R#, D/C#, M/IO#	HIGH	Bus Hold
LOCK#	LOW	Bus Hold
PLOCK#	LOW	Bus Hold
ADS#	LOW	Bus Hold
BLAST#	LOW	Bus Hold
PCHK#	LOW	
FERR#	LOW	
A2-A3	HIGH	Bus, Address Hold

Table 1.2. Input Pins

Name	Active Level	Synchronous/Asynchronous
CLK		
RESET	HIGH	Asynchronous
HOLD	HIGH	Synchronous
AHOLD	HIGH	Synchronous
EADS#	LOW	Synchronous
BOFF#	LOW	Synchronous
FLUSH#	LOW	Asynchronous
A20M#	LOW	Asynchronous
BS16#, BS8#	LOW	Synchronous
KEN#	LOW	Synchronous
RDY#	LOW	Synchronous
BRDY#	LOW	Synchronous
INTR	HIGH	Asynchronous
NMI	HIGH	Asynchronous
IGNNE#	LOW	Asynchronous

Table 1.3. Input/Output Pins

Name	Active Level	When Floated
D0-D31	HIGH	Bus Hold
DP0-DP3	HIGH	Bus Hold
A4-A31	HIGH	Bus, Address Hold

2.0 ARCHITECTURAL OVERVIEW

The 486 microprocessor is a 32-bit architecture with on-chip memory management, floating point and cache memory units.

The 486 microprocessor contains all the features of the 386™ microprocessor with enhancements to increase performance. The instruction set includes the complete 386 microprocessor instruction set along with extensions to serve new applications. The on-chip memory management unit (MMU) is completely compatible with the 386 microprocessor MMU. The 486 microprocessor brings the 387™ math coprocessor on-chip. All software written for the 386 microprocessor, 387 math coprocessor and previous members of the 86/87 architectural family will run on the 486 microprocessor without any modifications.

Several enhancements have been added to the 486 microprocessor to increase performance. On-chip cache memory allows frequently used data and code to be stored on-chip reducing accesses to the external bus. RISC design techniques have been used to reduce instruction cycle times. A burst bus feature enables fast cache fills. All of these features, combined, lead to performance greater than twice that of a 386 microprocessor.

The memory management unit (MMU) consists of a segmentation unit and a paging unit. Segmentation allows management of the logical address space by providing easy data and code relocatability and efficient sharing of global resources. The paging mechanism operates beneath segmentation and is transparent to the segmentation process. Paging is optional and can be disabled by system software. Each segment can be divided into one or more 4 Kbyte segments. To implement a virtual memory system, the 486 microprocessor supports full restartability for all page and segment faults.

Memory is organized into one or more variable length segments, each up to four gigabytes (2^{32} bytes) in size. A segment can have attributes associated with it which include its location, size, type (i.e., stack, code or data), and protection characteristics. Each task on a 486 microprocessor can have a maximum of 16,381 segments, each up to four gigabytes in size. Thus each task has a maximum of 64 terabytes (trillion bytes) of virtual memory.

The segmentation unit provides four-levels of protection for isolating and protecting applications and the operating system from each other. The hardware enforced protection allows the design of systems with a high degree of integrity.

The 486 microprocessor has two modes of operation: Real Address Mode (Real Mode) and Protected

Mode Virtual Address Mode (Protected Mode). In Real Mode the 486 microprocessor operates as a very fast 8086. Real Mode is required primarily to set up the processor for Protected Mode operation. Protected Mode provides access to the sophisticated memory management paging and privilege capabilities of the processor.

Within Protected Mode, software can perform a task switch to enter into tasks designated as Virtual 8086 Mode tasks. Each virtual 8086 task behaves with 8086 semantics, allowing 8086 software (an application program or an entire operating system) to execute.

The on-chip floating point unit operates in parallel with the arithmetic and logic unit and provides arithmetic instructions for a variety of numeric data types. It executes numerous built-in transcendental functions (e.g., tangent, sine, cosine, and log functions). The floating point unit fully conforms to the ANSI/IEEE standard 754-1985 for floating point arithmetic.

The on-chip cache is 8 Kbytes in size. It is 4-way set associative and follows a write-through policy. The on-chip cache includes features to provide flexibility in external memory system design. Individual pages can be designated as cacheable or non-cacheable by software or hardware. The cache can also be enabled and disabled by software or hardware.

Finally the 486 microprocessor has features to facilitate high performance hardware designs. The 1X clock eases high frequency board level designs. The burst bus feature enables fast cache fills. These features are described beginning in Section 6.

2.1 Register Set

The 486 microprocessor register set includes all the registers contained in the 386 microprocessor and the 387 math coprocessor. The register set can be split into the following categories:

- Base Architecture Registers
 - General Purpose Registers
 - Instruction Pointer
 - Flags Register
 - Segment Registers
- Systems Level Registers
 - Control Registers
 - System Address Registers

- Floating Point Registers
 - Data Registers
 - Tag Word
 - Status Word
 - Instruction and Data Pointers
 - Control Word
- Debug and Test Registers

The base architecture and floating point registers are accessible by the applications program. The system level registers are only accessible at privilege level 0 and are used by the systems level program. The debug and test registers are also only accessible at privilege level 0.

2.1.1 BASE ARCHITECTURE REGISTERS

Figure 2.1 shows the 486 microprocessor base architecture registers. The contents of these registers are task-specific and are automatically loaded with a new context upon a task switch operation.

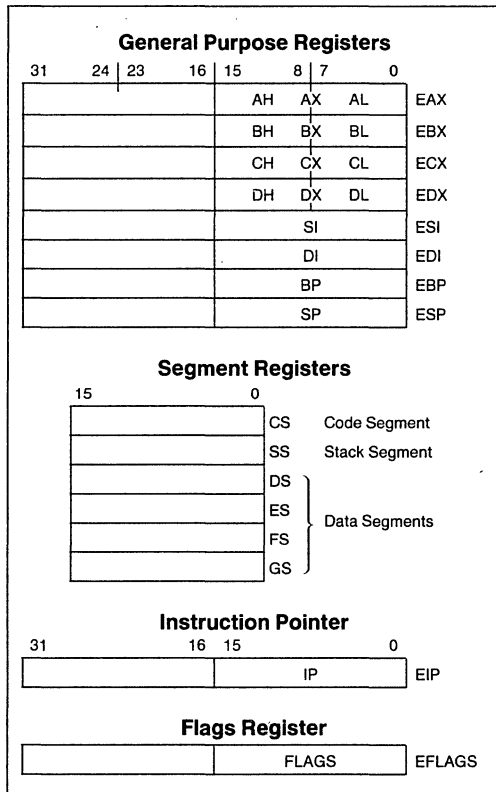


Figure 2.1. Base Architecture Registers

The base architecture includes six directly accessible descriptors, each specifying a segment up to 4 Gbytes in size. The descriptors are indicated by the selector values placed in the 486 microprocessor segment registers. Various selector values can be loaded as a program executes.

The selectors are also task-specific, so the segment registers are automatically loaded with new context upon a task switch operation.

2.1.1.1 General Purpose Registers

The eight 32-bit general purpose registers are shown in Figure 2.1. These registers hold data or address quantities. The general purpose registers can support data operands of 1, 8, 16 and 32 bits, and bit fields of 1 to 32 bits. Address operands of 16 and 32 bits are supported. The 32-bit registers are named EAX, EBX, ECX, EDX, ESI, EDI, EBP and ESP.

The least significant 16 bits of the general purpose registers can be accessed separately by using the 16-bit names of the registers AX, BX, CX, DX, SI, DI, BP and SP. The upper 16 bits of the register are not changed when the lower 16 bits are accessed separately.

Finally 8-bit operations can individually access the lowest byte (bits 0–7) and the higher byte (bits 8–15) of the general purpose registers AX, BX, CX and DX. The lowest bytes are named AL, BL, CL and DL respectively. The higher bytes are named AH, BH, CH and DH respectively. The individual byte accessibility offers additional flexibility for data operations but is not used for effective address calculation.

2.1.1.2 Instruction Pointer

The instruction pointer, shown in Figure 2.1, is a 32-bit register named EIP. EIP holds the offset of the next instruction to be executed. The offset is always relative to the base of the code segment (CS). The lower 16 bits (bits 0–15) of the EIP contain the 16-bit instruction pointer named IP, which is used for 16-bit addressing.

2.1.1.3 Flags Register

The flags register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS control certain operations and indicate status of the 486 microprocessor. The lower 16 bits (bit 0–15) of EFLAGS contain the 16-bit register named FLAGS, which is most useful when executing 8086 and 80286 code. EFLAGS is shown in Figure 2.2.

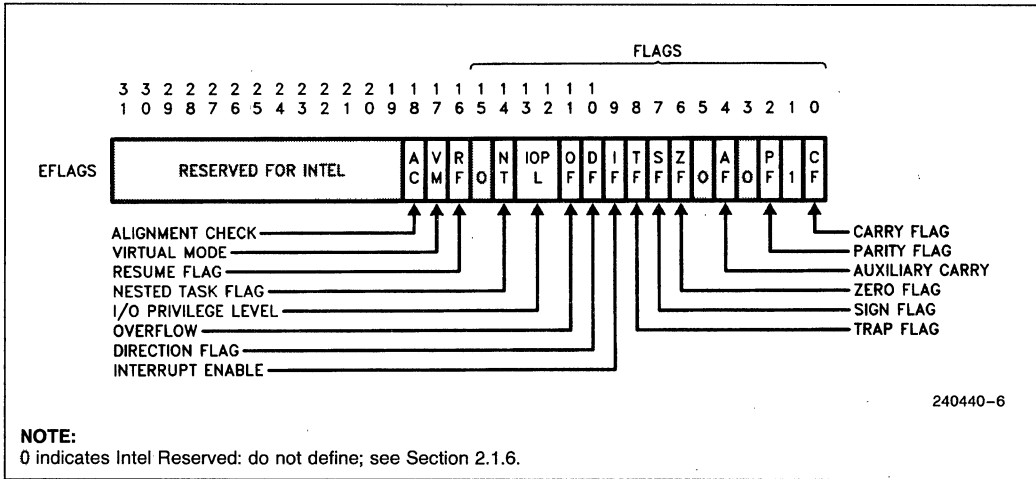


Figure 2.2. Flags Register

EFLAGS bits 1, 3, 5, 15 and 19–31 are “undefined”. When these bits are stored during interrupt processing or with a PUSHF instruction (push flags onto stack), a one is stored in bit 1 and zeros in bits 3, 5, 15 and 19–31.

The EFLAGS register in the 486 microprocessor contains a new bit not previously defined. The new bit, AC, is defined in the upper 16 bits of the register and it enables faults on accesses to misaligned data.

AC (Alignment Check, bit 18)

The AC bit enables the generation of faults if a memory reference is to a misaligned address. Alignment faults are enabled when AC is set to 1. A mis-aligned address is a word access

to an odd address, a dword access to an address that is not on a dword boundary, or an 8-byte reference to an address that is not on a 64-bit word boundary. See Section 7.1.6 for more information on operand alignment.

Alignment faults are only generated by programs running at privilege level 3. The AC bit setting is ignored at privilege levels 0, 1 and 2. Note that references to the descriptor tables (for selector loads), or the task state segment (TSS), are implicitly level 0 references even if the instructions causing the references are executed at level 3. Alignment faults are reported through interrupt 17, with an error code of 0. Table 2.1 gives the alignment required for the 486 microprocessor data types.

Table 2.1. Data Type Alignment Requirements

Memory Access	Alignment (Byte Boundary)
Word	2
Dword	4
Single Precision Real	4
Double Precision Real	8
Extended Precision Real	8
Selector	2
48-Bit Segmented Pointer	4
32-Bit Flat Pointer	4
32-Bit Segmented Pointer	2
48-Bit “Pseudo-Descriptor”	4
FSTENV/FLDENV Save Area	4/2 (On Operand Size)
FSAVE/FRSTOR Save Area	4/2 (On Operand Size)
Bit String	4

IMPLEMENTATION NOTE:

Several instructions on the 486 microprocessor generate misaligned references, even if their memory address is aligned. For example, on the 486 microprocessor, the SGDT/SIDT (store global/interrupt descriptor table) instruction reads/writes two bytes, and then reads/writes four bytes from a "pseudo-descriptor" at the given address. The 486 microprocessor will generate misaligned references unless the address is on a 2 mod 4 boundary. The FSAVE and FRSTOR instructions (floating point save and restore state) will generate misaligned references for one-half of the register save/restore cycles. The 486 microprocessor will not cause any AC faults if the effective address given in the instruction has the proper alignment.

VM (Virtual 8086 Mode, bit 17)

The VM bit provides Virtual 8086 Mode within Protected Mode. If set while the 486 Microprocessor is in Protected Mode, the 486 Microprocessor will switch to Virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes. The VM bit can be set only in Protected Mode, by the IRET instruction (if current privilege level = 0) and by task switches at any privilege level. The VM bit is unaffected by POPF. PUSHF always pushes a 0 in this bit, even if executing in Virtual 8086 Mode. The EFLAGS image pushed during interrupt processing or saved during task switches will contain a 1 in this bit if the interrupted code was executing as a Virtual 8086 Task.

RF (Resume Flag, bit 16)

The RF flag is used in conjunction with the debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. When RF is set, it causes any debug fault to be ignored on the next instruction. RF is then automatically reset at the successful completion of every instruction (no faults are signalled) except the IRET instruction, the POPF instruction, (and JMP, CALL, and INT instructions causing a task switch). These instructions set RF to the value specified by the memory image. For example, at the end of the breakpoint service routine, the IRET instruction can pop an EFLAG image having the RF bit set and resume the program's execution at the breakpoint address without generating another breakpoint fault on the same location.

NT (Nested Task, bit 14)

This flag applies to Protected Mode. NT is set to indicate that the execution of this task is nested within another task. If set, it indicates

that the current nested task's Task State Segment (TSS) has a valid back link to the previous task's TSS. This bit is set or reset by control transfers to other tasks. The value of NT in EFLAGS is tested by the IRET instruction to determine whether to do an inter-task return or an intra-task return. A POPF or an IRET instruction **will** affect the setting of this bit according to the image popped, at any privilege level.

IOPL (Input/Output Privilege Level, bits 12-13)

This two-bit field applies to Protected Mode. IOPL indicates the numerically maximum CPL (current privilege level) value permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O Permission Bitmap. It also indicates the maximum CPL value allowing alteration of the IF (INTR Enable Flag) bit when new values are popped into the EFLAGS register. POPF and IRET instruction can alter the IOPL field when executed at CPL = 0. Task switches can always alter the IOPL field, when the new flag image is loaded from the incoming task's TSS.

OF (Overflow Flag, bit 11)

OF is set if the operation resulted in a signed overflow. Signed overflow occurs when the operation resulted in carry/borrow **into** the sign bit (high-order bit) of the result but did not result in a carry/borrow **out of** the high-order bit, or vice-versa. For 8-, 16-, 32-bit operations, OF is set according to overflow at bit 7, 15, 31, respectively.

DF (Direction Flag, bit 10)

DF defines whether ESI and/or EDI registers postdecrement or postincrement during the string instructions. Postincrement occurs if DF is reset. Postdecrement occurs if DF is set.

IF (INTR Enable Flag, bit 9)

The IF flag, when set, allows recognition of external interrupts signalled on the INTR pin. When IF is reset, external interrupts signalled on the INTR are not recognized. IOPL indicates the maximum CPL value allowing alteration of the IF bit when new values are popped into EFLAGS or FLAGS.

TF (Trap Enable Flag, bit 8)

TF controls the generation of exception 1 trap when single-stepping through code. When TF is set, the 486 Microprocessor generates an exception 1 trap after the next instruction is executed. When TF is reset, exception 1 traps occur only as a function of the breakpoint addresses loaded into debug registers DR0-DR3.

- SF (Sign Flag, bit 7)
SF is set if the high-order bit of the result is set, it is reset otherwise. For 8-, 16-, 32-bit operations, SF reflects the state of bit 7, 15, 31 respectively.
- ZF (Zero Flag, bit 6)
ZF is set if all bits of the result are 0. Otherwise it is reset.
- AF (Auxiliary Carry Flag, bit 4)
The Auxiliary Flag is used to simplify the addition and subtraction of packed BCD quantities. AF is set if the operation resulted in a carry out of bit 3 (addition) or a borrow into bit 3 (subtraction). Otherwise AF is reset. AF is affected by carry out of, or borrow into bit 3 only, regardless of overall operand length: 8, 16 or 32 bits.
- PF (Parity Flags, bit 2)
PF is set if the low-order eight bits of the operation contains an even number of "1's" (even parity). PF is reset if the low-order eight bits have odd parity. PF is a function of only the low-order eight bits, regardless of operand size.
- CF (Carry Flag, bit 0)
CF is set if the operation resulted in a carry out of (addition), or a borrow into (subtraction) the high-order bit. Otherwise CF is reset. For 8-, 16- or 32-bit operations, CF is set according to carry/borrow at bit 7, 15 or 31, respectively.

NOTE:

In these descriptions, "set" means "set to 1," and "reset" means "reset to 0."

2.1.1.4 Segment Registers

Six 16-bit segment registers hold segment selector values identifying the currently addressable memory segments. In protected mode, each segment may range in size from one byte up to the entire linear and physical address space of the machine, 4 Gbytes (2^{32} bytes). In real address mode, the maximum segment size is fixed at 64 Kbytes (2^{16} bytes).

The six addressable segments are defined by the segment registers CS, SS, DS, ES, FS and GS. The selector in CS indicates the current code segment; the selector in SS indicates the current stack segment; the selectors in DS, ES, FS and GS indicate the current data segments.

2.1.1.5 Segment Descriptor Cache Registers

The segment descriptor cache registers are not programmer visible, yet it is very useful to understand their content. A programmer invisible descriptor cache register is associated with each programmer-visible segment register, as shown by Figure 2.3. Each descriptor cache register holds a 32-bit base address, a 32-bit segment limit, and the other necessary segment attributes.

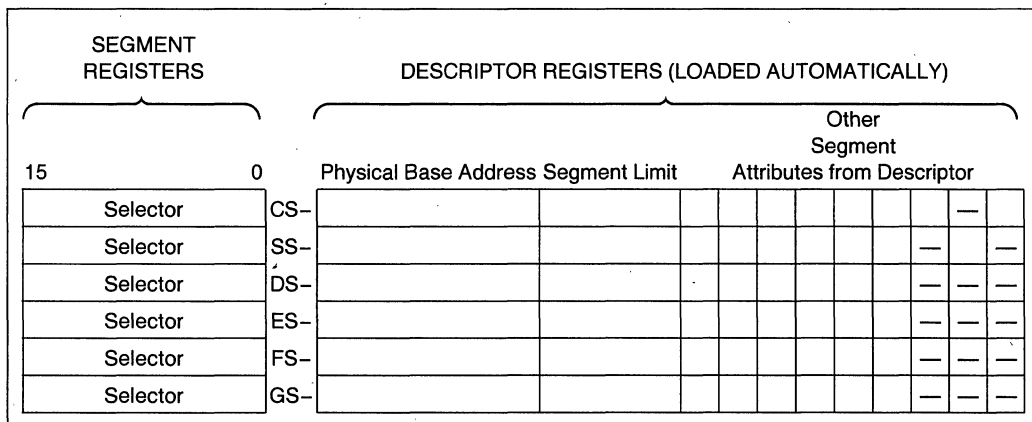


Figure 2.3. i486™ Microprocessor Segment Registers and Associated Descriptor Cache Registers

When a selector value is loaded into a segment register, the associated descriptor cache register is automatically updated with the correct information. In Real Address Mode, only the base address is updated directly (by shifting the selector value four bits to the left), since the segment maximum limit and attributes are fixed in Real Mode. In Protected Mode, the base address, the limit, and the attributes are all updated per the contents of the segment descriptor indexed by the selector.

Whenever a memory reference occurs, the segment descriptor cache register associated with the segment being used is automatically involved with the memory reference. The 32-bit segment base address becomes a component of the linear address calculation, the 32-bit limit is used for the limit-check operation, and the attributes are checked against the type of memory reference requested.

2.1.2 SYSTEM LEVEL REGISTERS

The system level registers, Figure 2.4, control operation of the on-chip cache, the on-chip floating point

unit (FPU) and the segmentation and paging mechanisms. These registers are only accessible to programs running at privilege level 0, the highest privilege level.

The system level registers include three control registers and four segmentation base registers. The three control registers are CR0, CR2 and CR3. CR1 is reserved for future Intel processors. The four segmentation base registers are the Global Descriptor Table Register (GDTR), the Interrupt Descriptor Table Register (GDTR), the Local Descriptor Table Register (LDTR) and the Task State Segment Register (TR).

2.1.2.1 Control Registers

Control Register 0 (CR0)

CR0, shown in Figure 2.5, contains 10 bits for control and status purposes. Five of the bits defined in the 486 microprocessor's CR0 are newly defined. The new bits are CD, NW, AM, WP and NE. The function of the bits in CR0 can be categorized as follows:

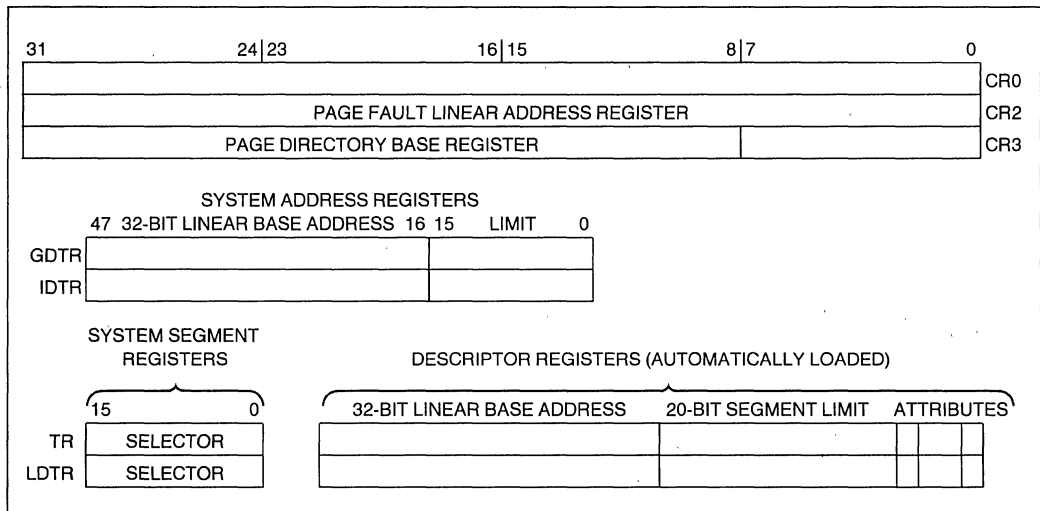


Figure 2.4. System Level Registers

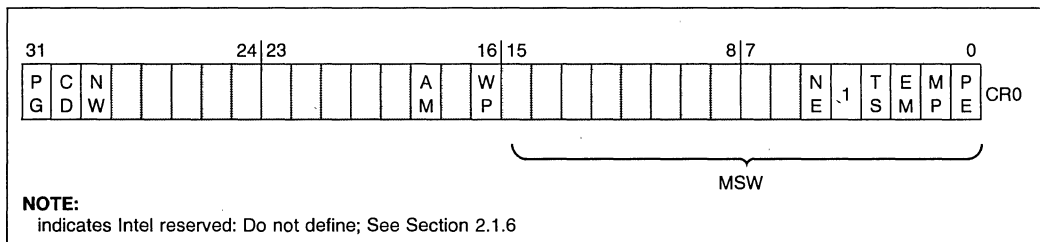


Figure 2.5. Control Register 0

486 Microprocessor Operating Modes: PG, PE (Table 2.2)

On-Chip Cache Control Modes: CD, NW (Table 2.3)

On-Floating Point Unit Control: TS, EM, MP, NE (Table 2.4)

Alignment Check Control: AM

Supervisor Write Protect: WP

Table 2.2. Processor Operating Modes

PG	PE	Mode
0	0	REAL Mode. Exact 8086 semantics, with 32-bit extensions available with prefixes.
0	1	Protected Mode. Exact 80286 semantics, plus 32-bit extensions through both prefixes and "default" prefix setting associated with code segment descriptors. Also, a sub-mode is defined to support a virtual 8086 within the context of the extended 80286 protection model.
1	0	UNDEFINED. Loading CR0 with this combination of PG and PE bits will raise a GP fault with error code 0.
1	1	Paged Protected Mode. All the facilities of Protected mode, with paging enabled underneath segmentation.

Table 2.3. On-Chip Cache Control Modes

CD	NW	Operating Mode
1	1	Cache fills disabled, write-through and invalidates disabled.
1	0	Cache fills disabled, write-through and invalidates enabled.
0	1	INVALID. If CR0 is loaded with this configuration of bits, a GP fault with error code is raised.
0	0	Cache fills enabled, write-through and invalidates enabled.

Table 2.4. On-Chip Floating Point Unit Control

CR0 BIT			Instruction Type	
EM	TS	MP	Floating-Point	Wait
0	0	0	Execute	Execute
0	0	1	Execute	Execute
0	1	0	Trap 7	Execute
0	1	1	Trap 7	Trap 7
1	0	0	Trap 7	Execute
1	0	1	Trap 7	Execute
1	1	0	Trap 7	Execute
1	1	1	Trap 7	Trap 7

The low-order 16 bits of CR0 are also known as the Machine Status Word (MSW), for compatibility with the 80286 protected mode. LMSW and SMSW (load and store MSW) instructions are taken as special aliases of the load and store CR0 operations, where only the low-order 16 bits of CR0 are involved. The LMSW and SMSW instructions in the 486 microprocessor work in an identical fashion to the LMSW and SMSW instructions in the 80286 (i.e., they only operate on the low-order 16 bits of CR0 and ignores the new bits). New 486 microprocessor operating systems should use the MOV CR0, Reg instruction.

The defined CR0 bits are described below.

PG (Paging Enable, bit 31)

The PG bit is used to indicate whether paging is enabled (PG=1) or disabled (PG=0). See Table 2.2.

CD (Cache Disable, bit 30)

The CD bit is used to enable the on-chip cache. When CD=1, the cache will not be filled on cache misses. When CD=0, cache fills may be performed on misses. See Table 2.3.

The state of the CD bit, the cache enable input pin (KEN#), and the relevant page cache disable (PCD) bit determine if a line read in response to a cache miss will be installed in the cache. A line is installed in the cache only if CD=0 and KEN# and PCD are both zero. The relevant PCD bit comes from either the page table entry, page directory entry or control register 3. Refer to Section 5.6 for more details on page cacheability.

CD is set to one after RESET.

NW (Not Write-Through, bit 29)

The NW bit enables on-chip cache write-throughs and write-invalidate cycles (NW=0). When NW=0, all writes, including cache hits, are sent out to the pins. Invalidate cycles are enabled when NW=0. During an invalidate cycle a line will be removed from the cache if the invalidate address hits in the cache. See Table 2.3.

When NW=1, write-throughs and write-invalidate cycles are disabled. A write will not be sent to the pins if the write hits in the cache. With NW=1 the only write cycles that reach the external bus are cache misses. Write hits with NW=1 will never update main memory. Invalidate cycles are ignored when NW=1.

AM (Alignment Mask, bit 18)

The AM bit controls whether the alignment check (AC) bit in the flag register (EFLAGS) can allow an alignment fault. AM=0 disables the AC bit. AM=1 enables the AC bit. AM=0 is the 386 microprocessor compatible mode.

386 microprocessor software may load incorrect data into the AC bit in the EFLAGS register. Setting AM=0 will prevent AC faults from occurring before the 486 microprocessor has created the AC interrupt service routine.

WP (Write Protect, bit 16)

WP protects read-only pages from supervisor write access. The 386 microprocessor allows a read-only page to be written from privilege levels 0–2. The 486 microprocessor is compatible with the 386 microprocessor when WP=0. WP=1 forces a fault on a write to a read-only page from any privilege level. Operating systems with Copy-on-Write features can be supported with the WP bit. Refer to Section 4.5.3 for further details on use of the WP bit.

NE (Numerics Exception, bit 5)

The NE bit controls whether unmasked floating point exceptions (UFPE) are handled through interrupt vector 16 (NE=1) or through an external interrupt (NE=0). NE=0 (default at reset) supports the DOS operating system error reporting scheme from the 8087, 80287 and 387 math coprocessor. In DOS systems, math coprocessor errors are reported via external interrupt vector 13. DOS uses interrupt vector 16 for an operating system call. Refer to Sections 6.2.13 and 7.2.14 for more information on floating point error reporting.

For any UFPE the floating point error output pin (FERR#) will be driven active.

For NE=0, the 486 microprocessor works in conjunction with the ignore numeric error input (IGNNE#) and the FERR# output pins. When a UFPE occurs and the IGNNE# input is inactive, the 486 microprocessor freezes immediately before executing the next floating point instruction. An external interrupt controller will supply an interrupt vector when FERR# is driven active. The UFPE is ignored if IGNNE# is active and floating point execution continues.

NOTE:

The freeze does not take place if the next instruction is one of the control instructions FNCLX, FNINIT, FNSAVE, FNSTENV, FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI and FNSETPM. The freeze does occur if the next instruction is WAIT.

For NE=1, any UFPE will result in a software interrupt 16, immediately before executing the next non-control floating point or WAIT instruction. The ignore numeric error input (IGNNE#) signal will be ignored.

TS (Task Switched, bit 3)

The TS bit is set whenever a task switch operation is performed. Execution of a floating point instruction with TS=1 will cause a device not available (DNA) fault (trap vector 7). If TS=1 and MP=1 (monitor coprocessor in CR0) a WAIT instruction will cause a DNA fault. See Table 2.4.

EM (Emulate Coprocessor, bit 2)

The EM bit determines whether floating point instructions are trapped (EM=1) or executed. If EM=1, all floating point instructions will cause fault 7.

NOTE:

WAIT instructions are not affected by the state of EM. See Table 2.4.

MP (Monitor Coprocessor, bit 1)

The MP bit is used in conjunction with the TS bit to determine if WAIT instructions should trap. If MP=1 and TS=1, WAIT instructions cause fault 7. Refer to Table 2.4. The TS bit is set to 1 on task switches by the 486 microprocessor. Floating point instructions are not affected by the state of the MP bit. It is recommended that the MP bit be set to one for the normal operation of the 486 microprocessor.

PE (Protection Enable, bit 0)

The PE bit enables the segment based protection mechanism. If PE=1 protection is enabled. When PE=0 the 486 microprocessor operates in REAL mode, with segment based protection disabled, and addresses formed as in an 8086. Refer to Table 2.2.

All new CR0 bits added to the 386 and 486 microprocessors, except for ET and NE, are upward compatible with the 80286 because they are in register bits not defined in the 80286. For strict compatibility with the 80286, the load machine status word (LMSW) instruction is defined to not change the ET or NE bits.

Control Register 1 (CR1)

CR1 is reserved for use in future Intel microprocessors.

Control Register 2 (CR2)

CR2, shown in Figure 2.6, holds the 32-bit linear address that caused the last page fault detected. The error code pushed onto the page fault handler's stack when it is invoked provides additional status information on this page fault.

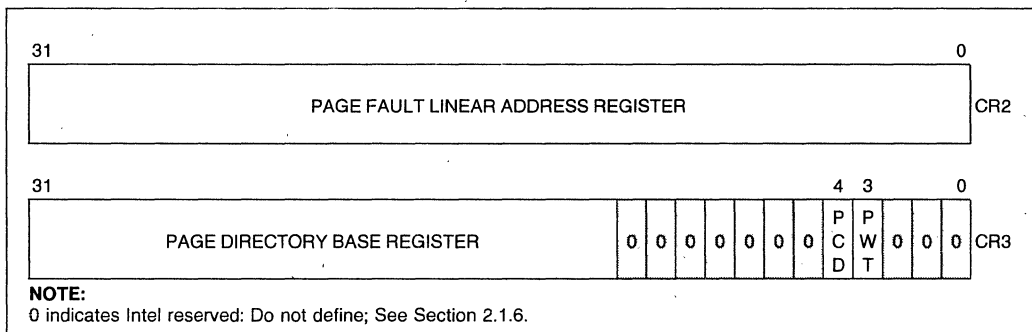


Figure 2.6. Control Registers 2 and 3

Control Register 3 (CR3)

CR3, shown in Figure 2.6, contains the physical base address of the page directory table. The 486 microprocessor page directory is always page aligned (4 Kbyte-aligned). This alignment is enforced by only storing bits 20–31 in CR3.

In the 486 microprocessor CR3 contains two new bits, page write-through (PWT) (bit 3) and page cache disable (PCD) (bit 4). The page table entry (PTE) and page directory entry (PDE) also contain PWT and PCD bits. PWT and PCD control page cacheability. When a page is accessed in external memory, the state of PWT and PCD are driven out on the PWT and PCD pins. The source of PWT and PCD can be CR3, the PTE or the PDE. PWT and PCD are sourced from CR3 under two conditions: when paging is disabled (PG = 0 in CR0) or when the PDE is being updated.

A task switch through a task state segment (TSS) which changes the values in CR3, or an explicit load into CR3 with any value, will invalidate all cached page table entries in the translation lookaside buffer (TLB).

The page directory base address in CR3 is a physical address. The page directory can be paged out while its associated task is suspended, but the operating system must ensure that the page directory is resident in physical memory before the task is dispatched. The entry in the TSS for CR3 has a physical address, with no provision for a present bit. This means that the page directory for a task must be resident in physical memory. The CR3 image in a TSS must point to this area, before the task can be dispatched through its TSS.

2.1.2.2 System Address Registers

Four special registers are defined to reference the tables or segments supported by the 80286, 386 and 486 microprocessor protection model. These tables or segments are:

- GDT (Global Descriptor Table)
- IDT (Interrupt Descriptor Table)
- LDT (Local Descriptor Table)
- TSS (Task State Segment)

The addresses of these tables and segments are stored in special registers, the System Address and System Segment Registers, illustrated in Figure 2.4. These registers are named GDTR, IDTR, LDTR and TR respectively. Section 4, Protected Mode Architecture, describes the use of these registers.

System Address Registers: GDTR and IDTR

The GDTR and IDTR hold the 32-bit linear base address and 16-bit limit of the GDT and IDT, respectively.

Since the GDT and IDT segments are global to all tasks in the system, the GDT and IDT are defined by 32-bit linear addresses (subject to page translation if paging is enabled) and 16-bit limit values.

System Segment Registers: LDTR and TR

The LDTR and TR hold the 16-bit selector for the LDT descriptor and the TSS descriptor, respectively.

Since the LDT and TSS segments are task specific segments, the LDT and TSS are defined by selector values stored in the system segment registers.

NOTE:

A programmer-invisible segment descriptor register is associated with each system segment register.

2.1.3 FLOATING POINT REGISTERS

Figure 2.7 shows the floating point register set. The on-chip FPU contains eight data registers, a tag word, a control register, a status register, an instruction pointer and a data pointer.

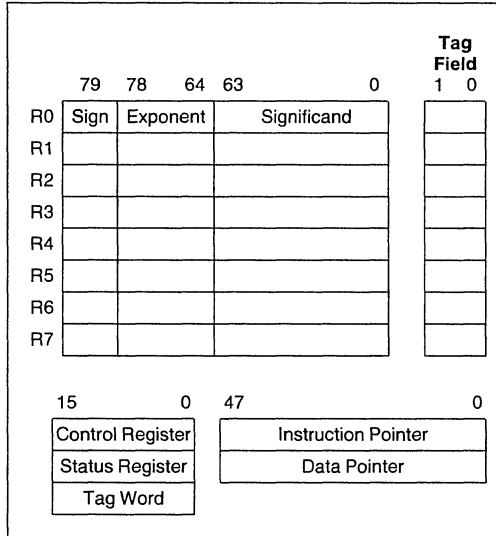


Figure 2.7. Floating Point Registers

The operation of the 486 microprocessor's on-chip floating point unit is exactly the same as the 387 math coprocessor. Software written for the 387 math coprocessor will run on the on-chip floating point unit (FPU) without any modifications.

2.1.3.1 Data Registers

Floating point computations use the 486 microprocessor's FPU data registers. These eight 80-bit registers provide the equivalent capacity of twenty 32-bit registers. Each of the eight data registers is divided

into "fields" corresponding to the FPU's extended-precision data type.

The FPU's register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers. The TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by one and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by one. Like other 486 microprocessor stacks in memory, the FPU register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to use. This explicit register addressing is also relative to TOP.

2.1.3.2 Tag Word

The tag word marks the content of each numeric data register, as shown in Figure 2.8. Each two-bit tag represents one of the eight data registers. The principal function of the tag word is to optimize the FPU's performance and stack handling by making it possible to distinguish between empty and nonempty register locations. It also enables exception handlers to check the contents of a stack location without the need to perform complex decoding of the actual data.

2.1.3.3 Status Word

The 16-bit status word reflects the overall state of the FPU. The status word is shown in Figure 2.9 and is located in the status register.

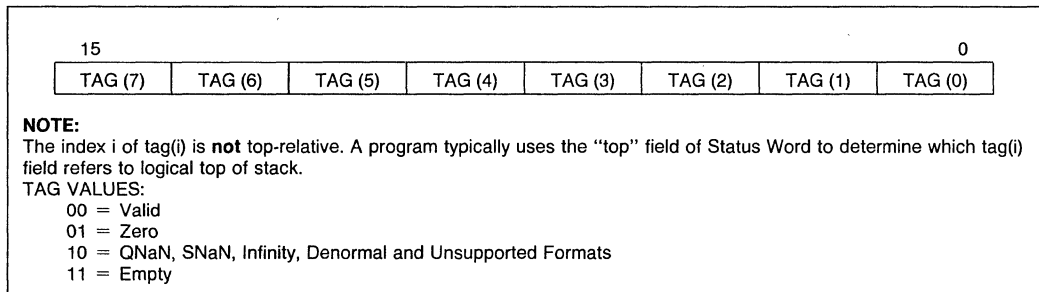


Figure 2.8. FPU Tag Word

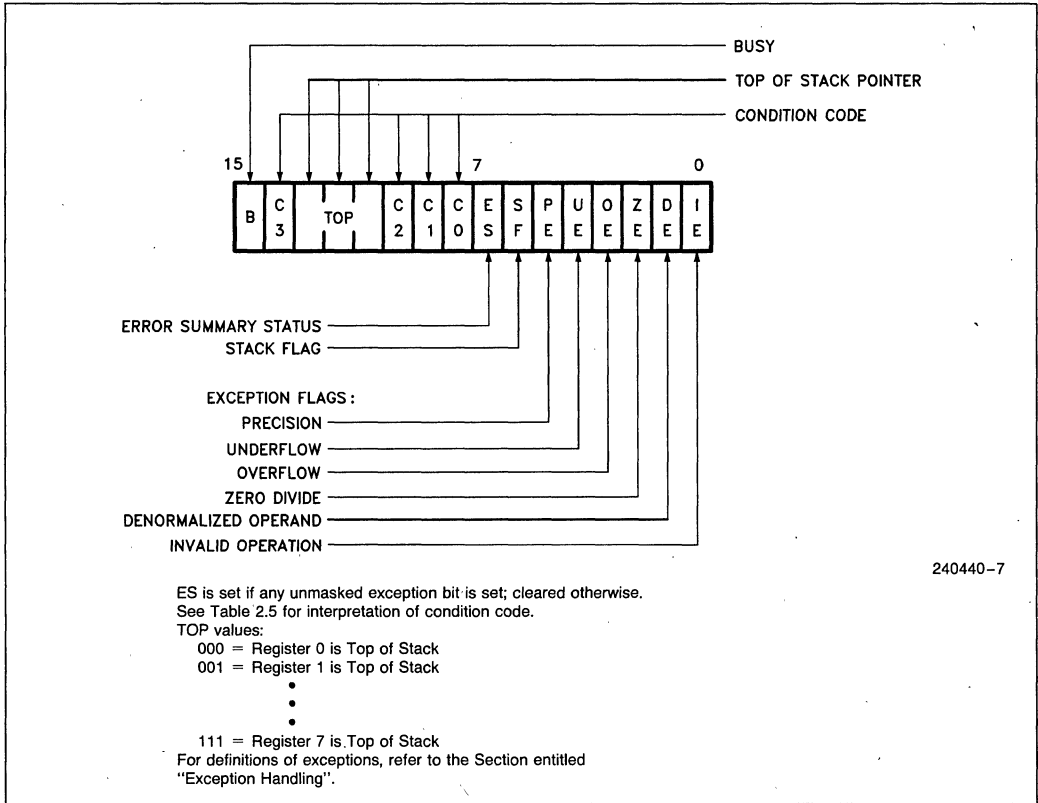


Figure 2.9. FPU Status Word

The B bit (Busy, bit 15) is included for 8087 compatibility. The B bit reflects the contents of the ES bit (bit 7 of the status word).

Bits 13–11 (TOP) point to the FPU register that is the current top-of-stack.

The four numeric condition code bits, C0–C3, are similar to the flags in EFLAGS. Instructions that perform arithmetic operations update C0–C3 to reflect the outcome. The effects of these instructions on the condition codes are summarized in Tables 2.5 through 2.8.

Table 2.5. FPU Condition Code Interpretation

Instruction	C0 (S)	C3 (Z)	C1 (A)	C2 (C)
FPREM, FPREM1 (see Table 2.3)	Three least significant bits of quotient Q2 Q0			Reduction 0 = complete 1 = incomplete
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 2.7)		Zero or O/U#	Operand is not comparable (Table 2.7)
FXAM	Operand class (see Table 2.8)		Sign or O/U#	Operand class (Table 2.8)
FCHS, FABS, FXCH, FINCTOP, FDECTOP, Constant loads, FXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/U#	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/U#	UNDEFINED
FPTAN, FSIN FCOS, FSINCOS	UNDEFINED		Roundup or O/U#, undefined if C2 = 1	Reduction 0 = complete 1 = incomplete
FLDENV, FRSTOR	Each bit loaded from memory			
FINIT	Clears these bits			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FSAVE	UNDEFINED			
O/U#	When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).			
Reduction	If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS, and FSINCOS, the reduction bit is set if the operand at the top of the stack is too large. In this case the original operand remains at the top of the stack.			
Roundup	When the PE bit of the status word is set, this bit indicates whether the last rounding in the instruction was upward.			
UNDEFINED	Do not rely on finding any specific value in these bits.			

Table 2.6. Condition Code Interpretation after FPREM and FPREM1 Instructions

Condition Code				Interpretation after FPREM and FPREM1	
C2	C3	C1	C0		
1	X	X	X	Incomplete Reduction: further interaction required for complete reduction	
0	Q1	Q0	Q2	Q MOD8	Complete Reduction: C0, C3, C1 contain three least significant bits of quotient
	0	0	0	0	
	0	1	0	1	
	1	0	0	2	
	1	1	0	3	
	0	0	1	4	
	0	1	1	5	
	1	0	1	6	
1	1	1	7		

Table 2.7. Condition Code Resulting from Comparison

Order	C3	C2	C0
TOP > Operand	0	0	0
TOP < Operand	0	0	1
TOP = Operand	1	0	0
Unordered	1	1	1

Table 2.8. Condition Code Defining Operand Class

C3	C2	C1	C0	Value at TOP
0	0	0	0	+ Unsupported
0	0	0	1	+ NaN
0	0	1	0	- Unsupported
0	0	1	1	- NaN
0	1	0	0	+ Normal
0	1	0	1	+ Infinity
0	1	1	0	- Normal
0	1	1	1	- Infinity
1	0	0	0	+ 0
1	0	0	1	+ Empty
1	0	1	0	- 0
1	0	1	1	- Empty
1	1	0	0	+ Denormal
1	1	1	0	- Denormal

Bit 7 is the error summary (ES) status bit. The ES bit is set if any unmasked exception bit (bits 0–5 in the status word) is set; ES is clear otherwise. The FERR# (floating point error) signal is asserted when ES is set.

Bit 6 is the stack flag (SF). This bit is used to distinguish invalid operations due to stack overflow or underflow. When SF is set, bit 9 (C1) distinguishes between stack overflow (C1=1) and underflow (C1=0).

Table 2.9 shows the six exception flags in bits 0–5 of the status word. Bits 0–5 are set to indicate that the FPU has detected an exception while executing an instruction.

The six exception flags in the status word can be individually masked by mask bits in the FPU control word. Table 2.9 lists the exception conditions, and their causes in order of precedence. Table 2.9 also shows the action taken by the FPU if the corresponding exception flag is masked.

An exception that is not masked by the control word will cause three things to happen: the corresponding exception flag in the status word will be set, the ES bit in the status word will be set and the FERR# output signal will be asserted. When the 486 microprocessor attempts to execute another floating point or WAIT instruction, exception 16 occurs or an external interrupt happens if the NE = 1 in control register

0. The exception condition must be resolved via an interrupt service routine. The FPU saves the address of the floating point instruction that caused the exception and the address of any memory operand required by that instruction in the instruction and data pointers (see Section 2.1.3.4).

Note that when a new value is loaded into the status word by the FLDENV (load environment) or FRSTOR (restore state) instruction, the value of ES (bit 7) and its reflection in the B bit (bit 15) are not derived from the values loaded from memory. The values of ES and B are dependent upon the values of the exception flags in the status word and their corresponding masks in the control word. If ES is set in such a case, the FERR# output of the 486 microprocessor is activated immediately.

2.1.3.4 Instruction and Data Pointers

Because the FPU operates in parallel with the ALU (in the 486 microprocessor the arithmetic and logic unit (ALU) consists of the base architecture registers), any errors detected by the FPU may be reported after the ALU has executed the floating point instruction that caused it. To allow identification of the failing numeric instruction, the 486 microprocessor contains two pointer registers that supply the address of the failing numeric instruction and the address of its numeric memory operand (if appropriate).

Table 2.9. FPU Exceptions

Exception	Cause	Default Action (if exception is masked)
Invalid Operation	Operation on a signaling NaN, unsupported format, indeterminate form ($0 \cdot \infty$, $0/0$, $(+\infty) + (-\infty)$, etc.), or stack overflow/underflow (SF is also set).	Result is a quiet NaN, integer indefinite, or BCD indefinite
Denormalized Operand	At least one of the operands is denormalized, i.e., it has the smallest exponent but a nonzero significand.	Normal processing continues
Zero Divisor	The divisor is zero while the dividend is a noninfinite, nonzero number.	Result is ∞
Overflow	The result is too large in magnitude to fit in the specified format.	Result is largest finite value or ∞
Underflow	The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes loss of accuracy.	Result is denormalized or zero
Inexact Result (Precision)	The true result is not exactly representable in the specified format (e.g., $1/3$); the result is rounded according to the rounding mode.	Normal processing continues

The instruction and data pointers are provided for user-written error handlers. These registers are accessed by the FLDENV (load environment), FSTENV (store environment), FSAVE (save state) and FRSTOR (restore state) instructions. Whenever the 486 microprocessor decodes a new floating point instruction, it saves the instruction (including any prefixes that may be present), the address of the operand (if present) and the opcode.

The instruction and data pointers appear in one of four formats depending on the operating mode of the 486 microprocessor (protected mode or real-ad-

dress mode) and depending on the operand-size attribute in effect (32-bit operand or 16-bit operand). When the 486 microprocessor is in the virtual-86 mode, the real address mode formats are used. The four formats are shown in Figures 2.10–2.13. The floating point instructions FLDENV, FSTENV, FSAVE and FRSTOR are used to transfer these values to and from memory. Note that the value of the data pointer is undefined if the prior floating point instruction did not have a memory operand.

NOTE:

The operand size attribute is the D bit in a segment descriptor.

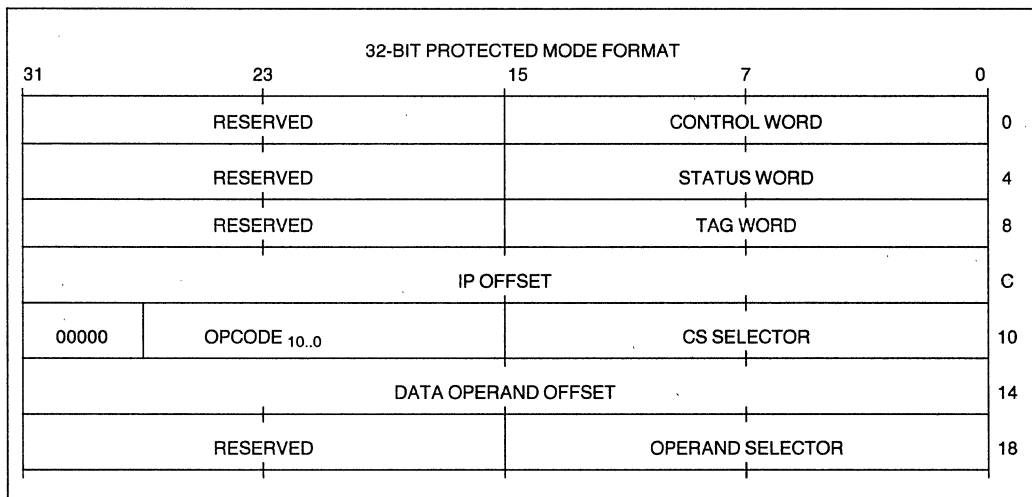


Figure 2.10. Protected Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

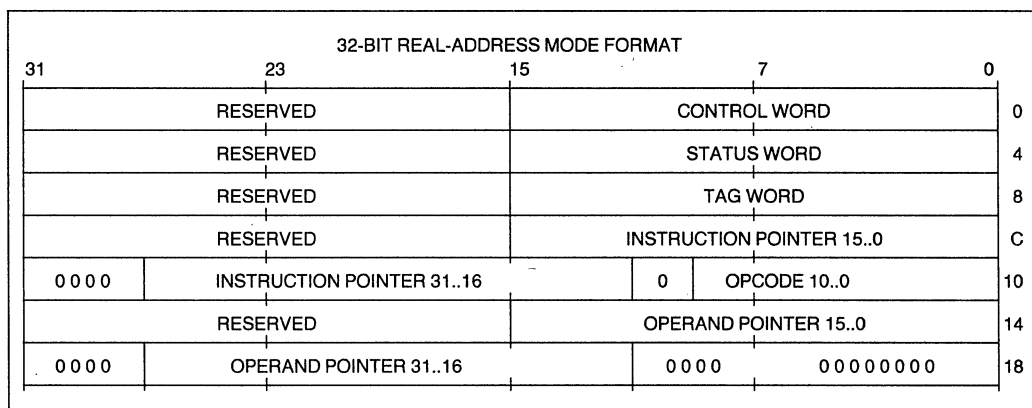


Figure 2.11. Real Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

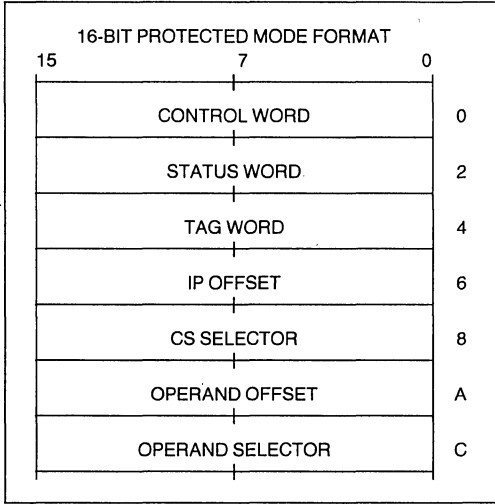


Figure 2.12. Protected Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

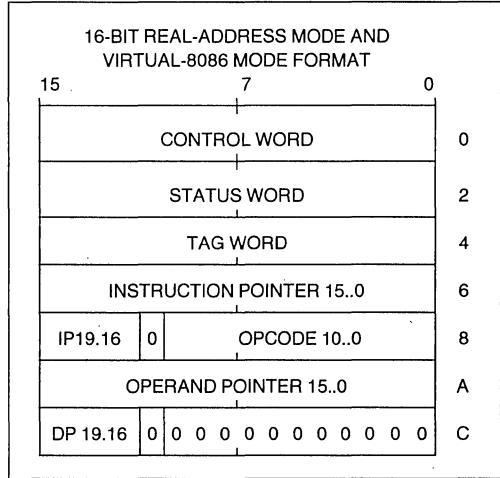


Figure 2.13. Real Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

2.1.3.5 FPU Control Word

The FPU provides several processing options that are selected by loading a control word from memory into the control register. Figure 2.14 shows the format and encoding of fields in the control word.

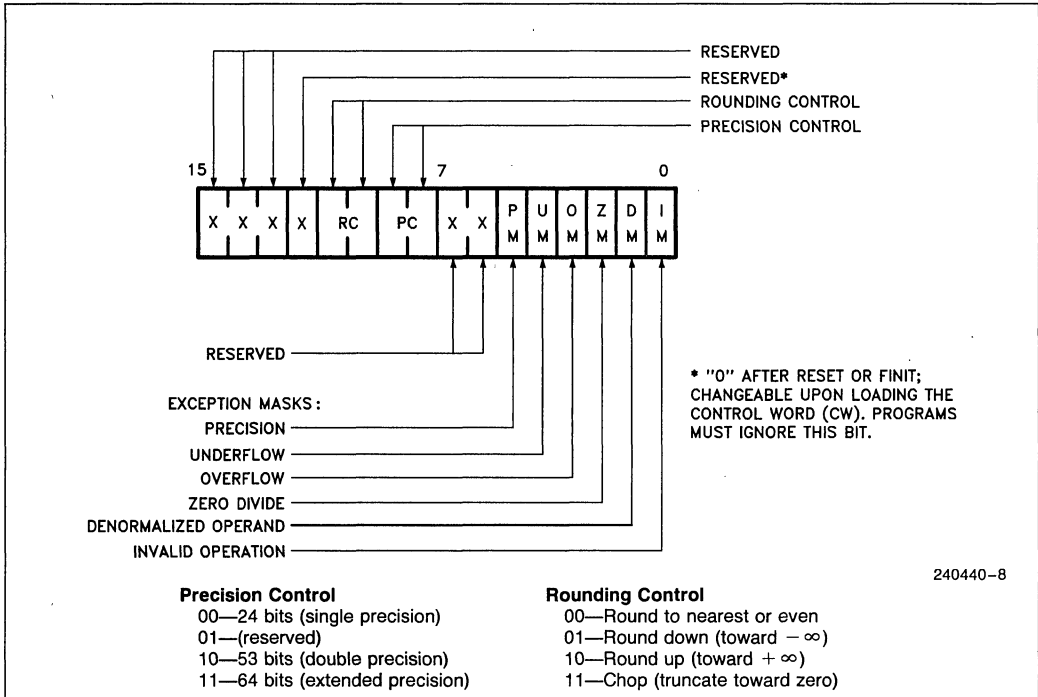


Figure 2.14. FPU Control Word

The low-order byte of the FPU control word configures the FPU error and exception masking. Bits 0–5 of the control word contain individual masks for each of the six exceptions that the FPU recognizes.

The high-order byte of the control word configures the FPU operating mode, including precision and rounding.

RC (Rounding Control, bits 10–11)

The RC bits provide for directed rounding and true chop, as well as the unbiased round to nearest even mode specified in the IEEE standard. Rounding control affects only those instructions that perform rounding at the end of the operation (and thus can generate a precision exception); namely, FST, FSTP, FIST, all arithmetic instructions (except FPREM, FPREM1, FXTRACT, FABS and FCHS), and all transcendental instructions.

PC (Precision Control, bits 8–9)

The PC bits can be used to set the FPU internal operating precision of the significand at less than the default of 64 bits (extended precision). This can be useful in providing compatibility with early generation arithmetic processors of smaller precision. PC affects only the instructions ADD, SUB, DIV, MUL, and SQRT. For all other instructions, either the precision is determined by the opcode or extended precision is used.

2.1.4 DEBUG AND TEST REGISTERS

2.1.4.1 Debug Registers

The six programmer accessible debug registers, Figure 2.15, provide on-chip support for debugging. Debug registers DR0–3 specify the four linear breakpoints. The Debug control register DR7, is used to set the breakpoints and the Debug Status Register, DR6, displays the current state of the breakpoints. The use of the Debug registers is described in Section 9.

Debug Registers	
LINEAR BREAKPOINT ADDRESS 0	DR0
LINEAR BREAKPOINT ADDRESS 1	DR1
LINEAR BREAKPOINT ADDRESS 2	DR2
LINEAR BREAKPOINT ADDRESS 3	DR3
Intel Reserved Do Not Define	DR4
Intel Reserved Do Not Define	DR5
BREAKPOINT STATUS	DR6
BREAKPOINT CONTROL	DR7
Test Registers	
CACHE TEST DATA	TR3
CACHE TEST STATUS	TR4
CACHE TEST CONTROL	TR5
TLB TEST CONTROL	TR6
TLB TEST STATUS	TR7

TLB = Translation Lookaside Buffer

Figure 2.15

2.1.4.2 Test Registers

The 486 microprocessor contains five test registers. The test registers are shown in Figure 2.15. TR6 and TR7 are used to control the testing of the translation lookaside buffer. TR3, TR4 and TR5 are used for testing the on-chip cache. The use of the test registers is discussed in Section 8.

2.1.5 REGISTER ACCESSIBILITY

There are a few differences regarding the accessibility of the registers in Real and Protected Mode. Table 2.10 summarizes these differences. See Section 4, Protected Mode Architecture, for further details.

Table 2.10. Register Usage

Register	Use in Real Mode		Use in Protected Mode		Use in Virtual 8086 Mode	
	Load	Store	Load	Store	Load	Store
General Registers	Yes	Yes	Yes	Yes	Yes	Yes
Segment Register	Yes	Yes	Yes	Yes	Yes	Yes
Flag Register	Yes	Yes	Yes	Yes	IOPL	IOPL*
Control Registers	Yes	Yes	PL = 0	PL = 0	No	Yes
GDTR	Yes	Yes	PL = 0	Yes	No	Yes
IDTR	Yes	Yes	PL = 0	Yes	No	Yes
LDTR	No	No	PL = 0	Yes	No	No
TR	No	No	PL = 0	Yes	No	No
FPU Data Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Control Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Status Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Instruction Pointer	Yes	Yes	Yes	Yes	Yes	Yes
FPU Data Pointer	Yes	Yes	Yes	Yes	Yes	Yes
Debug Registers	Yes	Yes	PL = 0	PL = 0	No	No
Test Registers	Yes	Yes	PL = 0	PL = 0	No	No

NOTES:

PL = 0: The registers can be accessed only when the current privilege level is zero.

*IOPL: The PUSHF and POPF instructions are made I/O Privilege Level sensitive in Virtual 86 Mode.

2.1.6 COMPATIBILITY

VERY IMPORTANT NOTE:

COMPATIBILITY WITH FUTURE PROCESSORS

In the preceding register descriptions, note certain 486 Microprocessor register bits are Intel reserved. When reserved bits are called out, treat them as fully undefined. This is essential for your software compatibility with future processors! Follow the guidelines below:

- 1) Do not depend on the states of any undefined bits when testing the values of defined register bits. Mask them out when testing.
- 2) Do not depend on the states of any undefined bits when storing them to memory or another register.

- 3) Do not depend on the ability to retain information written into any undefined bits.
- 4) When loading registers always load the undefined bits as zeros.
- 5) However, registers which have been previously stored may be reloaded without masking.

Depending upon the values of undefined register bits will make your software dependent upon the unspecified 486 Microprocessor handling of these bits. Depending on undefined values risks making your software incompatible with future processors that define usages for the 486 Microprocessor-undefined bits. **AVOID ANY SOFTWARE DEPENDENCE UPON THE STATE OF UNDEFINED 486 MICROPROCESSOR REGISTER BITS.**

2.2 Instruction Set

The 486 microprocessor instruction set can be divided into 11 categories of operations:

- Data Transfer
- Arithmetic
- Shift/Rotate
- String Manipulation
- Bit Manipulation
- Control Transfer
- High Level Language Support
- Operating System Support
- Processor Control
- Floating Point
- Floating Point Control

The 486 microprocessor instructions are listed in Section 10. Note that all floating point unit instruction mnemonics begin with an F.

All 486 microprocessor instructions operate on either 0, 1, 2 or 3 operands; where an operand resides in a register, in the instruction itself or in memory. Most zero operand instructions (e.g., CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the 486 microprocessor has a 32-byte instruction queue, an average of 10 instructions will be prefetched. The use of two operands permits the following types of common instructions:

- Register to Register
- Memory to Register
- Memory to Memory
- Immediate to Register
- Register to Memory
- Immediate to Memory

The operands can be either 8, 16, or 32 bits long. As a general rule, when executing code written for the 486 or 386 microprocessors (32-bit code), operands are 8 or 32 bits; when executing existing 80286 or 8086 code (16-bit code), operands are 8 or 16 bits. Prefixes can be added to all instructions which override the default length of the operands (i.e., use 32-bit operands for 16-bit code, or 16-bit operands for 32-bit code).

2.3 Memory Organization

Introduction

Memory on the 486 Microprocessor is divided up into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address, the high order byte at the

high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address, the high-order byte at the highest address. The address of a word or dword is the byte address of the low-order byte.

In addition to these basic data types, the 486 Microprocessor supports two larger units of memory: pages and segments. Memory can be divided up into one or more variable length segments, which can be swapped to disk or shared between programs. Memory can also be organized into one or more 4 Kbyte pages. Finally, both segmentation and paging can be combined, gaining the advantages of both systems. The 486 Microprocessor supports both pages and segments in order to provide maximum flexibility to the system designer. Segmentation and paging are complementary. Segmentation is useful for organizing memory in logical modules, and as such is a tool for the application programmer, while pages are useful for the system programmer for managing the physical memory of a system.

2.3.1 ADDRESS SPACES

The 486 Microprocessor has three distinct address spaces: **logical**, **linear**, and **physical**. A **logical** address (also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, DISPLACEMENT) discussed in Section 2.5.3 **Memory Addressing Modes** into an effective address. Since each task on the 486 Microprocessor has a maximum of 16K ($2^{14} - 1$) selectors, and offsets can be 4 gigabytes, (2^{32} bits) this gives a total of 2^{46} bits or 64 terabytes of **logical** address space per task. The programmer sees this virtual address space.

The segmentation unit translates the **logical** address space into a 32-bit **linear** address space. If the paging unit is not enabled then the 32-bit **linear** address corresponds to the **physical** address. The paging unit translates the **linear** address space into the **physical** address space. The **physical address** is what appears on the address pins.

The primary difference between Real Mode and Protected Mode is how the segmentation unit performs the translation of the **logical** address into the **linear** address. In Real Mode, the segmentation unit shifts the selector left four bits and adds the result to the offset to form the **linear** address. While in Protected Mode every selector has a **linear** base address associated with it. The **linear base** address is stored in one of two operating system tables (i.e., the Local Descriptor Table or Global Descriptor Table). The selector's **linear base** address is added to the offset to form the final **linear** address.

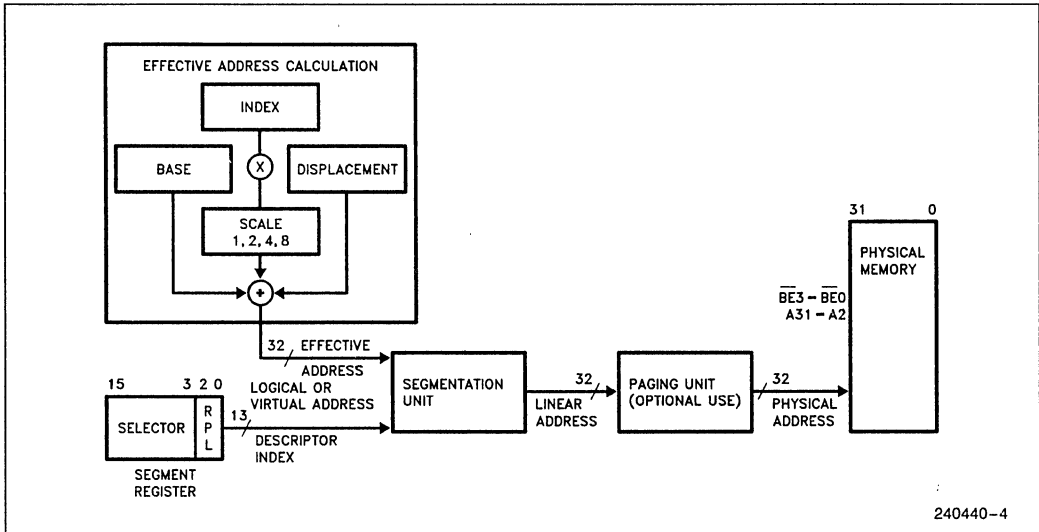


Figure 2.16. Address Translation

Figure 2.16 shows the relationship between the various address spaces.

2.3.2 SEGMENT REGISTER USAGE

The main data structure used to organize memory is the segment. On the 486 Microprocessor, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments: code and data, the segments are of variable size and can be as small as 1 byte or as large as 4 gigabytes (2^{32} bytes).

In order to provide compact instruction encoding, and increase processor performance, instructions do not need to explicitly specify which segment register is used. A default segment register is automatically chosen according to the rules of Table 2.11 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register; Stack references use the SS register and Instruction fetches use the CS register. The contents of the Instruction Pointer provide the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2.11. The override prefixes also allow the use of the ES, FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero

and create a system with a four gigabyte linear address space. This creates a system where the virtual address space is the same as the linear address space. Further details of segmentation are discussed in Section 4.1.

2.4 I/O Space

The 486 Microprocessor has two distinct physical address spaces: Memory and I/O. Generally, peripherals are placed in I/O space although the 486 Microprocessor also supports memory-mapped peripherals. The I/O space consists of 64 Kbytes, it can be divided into 64K 8-bit ports, 32K 16-bit ports, or 16K 32-bit ports, or any combination of ports which add up to less than 64 Kbytes. The 64K I/O address space refers to physical memory rather than linear address since I/O instructions do not go through the segmentation or paging hardware. The M/IO# pin acts as an additional address line thus allowing the system designer to easily determine which address space the processor is accessing.

The I/O ports are accessed via the IN and OUT I/O instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8- and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/IO# pin to be driven low.

I/O port addresses 00F8H through 00FFH are reserved for use by Intel.

Table 2.11. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA Instructions	SS	None
Source of POP, POPA, POPF, IRET, RET instructions	SS	None
Destination of STOS, MOVS, REP STOS, REP MOVS Instructions (DI is Base Register)	ES	None
Other Data References, with Effective Address Using Base Register of:		
[EAX]	DS	All
[EBX]	DS	
[ECX]	DS	
[EDX]	DS	
[ESI]	DS	
[EDI]	DS	
[EBP]	SS	
[ESP]	SS	

2.5 Addressing Modes

2.5.1 ADDRESSING MODES OVERVIEW

The 486 Microprocessor provides a total of 11 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

2.5.2 REGISTER AND IMMEDIATE MODES

Two of the addressing modes provide for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8-, 16- or 32-bit general registers.

Immediate Operand Mode: The operand is included in the instruction as part of the opcode.

2.5.3 32-BIT MEMORY ADDRESSING MODES

The remaining 9 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the segment base address and an effective address. The effective address is calculated by using combinations of the following four address elements:

DISPLACEMENT: An 8-, or 32-bit immediate value, following the instruction.

BASE: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area.

INDEX: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters.

SCALE: The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. Scaled index

mode is especially useful for accessing arrays or structures.

Combinations of these 4 components make up the 9 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of Base and Index components which requires one additional clock.

As shown in Figure 2.17, the effective address (EA) of an operand is calculated according to the following formula.

$$EA = \text{Base Reg} + (\text{Index Reg} * \text{Scaling}) + \text{Displacement}$$

Direct Mode: The operand's offset is contained as part of the instruction as an 8-, 16- or 32-bit displacement.

EXAMPLE: `INC Word PTR [500]`

Register Indirect Mode: A BASE register contains the address of the operand.

EXAMPLE: `MOV [ECX], EDX`

Based Mode: A BASE register's contents is added to a DISPLACEMENT to form the operand's offset.

EXAMPLE: `MOV ECX, [EAX + 24]`

Index Mode: An INDEX register's contents is added to a DISPLACEMENT to form the operand's offset.

EXAMPLE: `ADD EAX, TABLE[ESI]`

Scaled Index Mode: An INDEX register's contents is multiplied by a scaling factor which is added to a DISPLACEMENT to form the operand's offset.

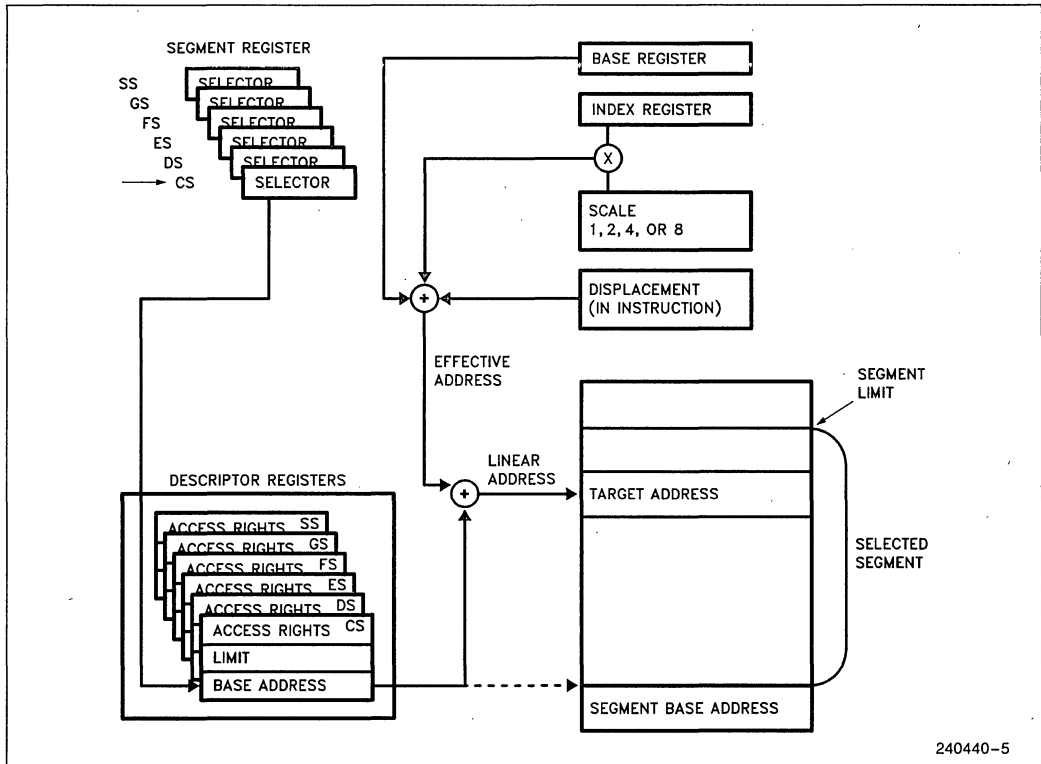
EXAMPLE: `IMUL EBX, TABLE[ESI*4],7`

Based Index Mode: The contents of a BASE register is added to the contents of an INDEX register to form the effective address of an operand.

EXAMPLE: `MOV EAX, [ESI] [EBX]`

Based Scaled Index Mode: The contents of an INDEX register is multiplied by a SCALING factor and the result is added to the contents of a BASE register to obtain the operand's offset.

EXAMPLE: `MOV ECX, [EDX*8] [EAX]`



240440-5

Figure 2.17. Addressing Mode Calculations

Based Index Mode with Displacement: The contents of an INDEX Register and a BASE register's contents and a DISPLACEMENT are all summed together to form the operand offset.

EXAMPLE: ADD EDX, [ESI] [EBP+00FFFFFF0H]

Based Scaled Index Mode with Displacement: The contents of an INDEX register are multiplied by a SCALING factor, the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.

EXAMPLE: MOV EAX, LOCALTABLE[EDI*4] [EBP+80]

2.5.4 DIFFERENCES BETWEEN 16- AND 32-BIT ADDRESSES

In order to provide software compatibility with the 80286 and the 8086, the 486 Microprocessor can execute 16-bit instructions in Real and Protected Modes. The processor determines the size of the instructions it is executing by examining the D bit in the CS segment Descriptor. If the D bit is 0 then all operand lengths and effective addresses are assumed to be 16 bits long. If the D bit is 1 then the default length for operands and addresses is 32 bits. In Real Mode the default size for operands and addresses is 16-bits.

Regardless of the default precision of the operands or addresses, the 486 Microprocessor is able to execute either 16- or 32-bit instructions. This is specified via the use of override prefixes. Two prefixes, the **Operand Size Prefix** and the **Address Length Prefix**, override the value of the D bit on an individual instruction basis. These prefixes are automatically added by Intel assemblers.

Example: The processor is executing in Real Mode and the programmer needs to access the EAX registers. The assembler code for this might be MOV EAX, 32-bit MEMORYOP, ASM486 Macro Assembler automatically determines that an Operand Size Prefix is needed and generates it.

Example: The D bit is 0, and the programmer wishes to use Scaled Index addressing mode to access an array. The Address Length Prefix allows the use of MOV DX, TABLE[ESI*2]. The assembler uses an

Address Length Prefix since, with D=0, the default addressing mode is 16-bits.

Example: The D bit is 1, and the program wants to store a 16-bit quantity. The Operand Length Prefix is used to specify only a 16-bit value; MOV MEM16, DX.

The OPERAND LENGTH and Address Length Prefixes can be applied separately or in combination to any instruction. The Address Length Prefix does not allow addresses over 64 Kbytes to be accessed in Real Mode. A memory address which exceeds FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional 486 Microprocessor addressing modes.

When executing 32-bit code, the 486 Microprocessor uses either 8-, or 32-bit displacements, and any register can be used as base or index registers. When executing 16-bit code, the displacements are either 8, or 16 bits, and the base and index register conform to the 80286 model. Table 2.12 illustrates the differences.

2.6 Data Formats

2.6.1 DATA TYPES

The 486 microprocessor can support a wide-variety of data types. In the following descriptions, the on-chip floating point unit (FPU) consists of the floating point registers. The central processing unit (CPU) consists of the base architecture registers.

2.6.1.1 Unsigned Data Types

The FPU does not support unsigned data types. Refer to Table 2.13.

Byte: Unsigned 8-bit quantity

Word: Unsigned 16-bit quantity

Dword: Unsigned 32-bit quantity

The least significant bit (LSB) in a byte is bit 0, and the most significant bit is 7.

Table 2.12. BASE and INDEX Registers for 16- and 32-Bit Addresses

	16-Bit Addressing	32-Bit Addressing
BASE REGISTER	BX, BP	Any 32-bit GP Register
INDEX REGISTER	SI, DI	Any 32-bit GP Register Except ESP
SCALE FACTOR	none	1, 2, 4, 8
DISPLACEMENT	0, 8, 16 bits	0, 8, 32 bits

2.6.1.2 Signed Data Types

All signed data types assume 2's complement notation. The signed data types contain two fields, a sign bit and a magnitude. The sign bit is the most significant bit (MSB). The number is negative if the sign bit is 1. If the sign bit is 0, the number is positive. The magnitude field consists of the remaining bits in the number. Refer to Table 2.13.

8-bit Integer: Signed 8-bit quantity

16-bit Integer: Signed 16-bit quantity

32-bit Integer: Signed 32-bit quantity

64-bit Integer: Signed 64-bit quantity

The FPU only supports 16-, 32- and 64-bit integers. The CPU only supports 8-, 16- and 32-bit integers.

2.6.1.3 Floating Point Data Types

Floating point data type in the 486 microprocessor contain three fields, sign, significand and exponent. The sign field is one bit and is the MSB of the floating point number. The number is negative if the sign bit is 1. If the sign bit is 0, the number is positive. The significand gives the significant bits of the number. The exponent field contains the power of 2 needed to scale the significand. Refer to Table 2.13.

Only the FPU supports floating point data types.

Single Precision Real: 23-bit significand and 8-bit exponent. 32 bits total.

Double Precision Real: 52-bit significand and 11-bit exponent. 64 bits total.

Extended Precision Real: 64-bit significand and 15-bit exponent. 80 bits total.

2.6.1.4 BCD Data Types

The 486 microprocessor supports packed and unpacked binary coded decimal (BCD) data types. A packed BCD data type contains two digits per byte, the lower digit is in bits 0–3 and the upper digit in bits 4–7. An unpacked BCD data type contains 1 digit per byte stored in bits 0–3.

The CPU supports 8-bit packed and unpacked BCD data types. The FPU only supports 80-bit packed BCD data types. Refer to Table 2.13.

2.6.1.5 String Data Types

A string data type is a contiguous sequence of bits, bytes, words or dwords. A string may contain between 1 byte and 4 Gbytes. Refer to Table 2.14.

String data types are only supported by the CPU.

Byte String: Contiguous sequence of bytes.

Word String: Contiguous sequence of words.

Dword String: Contiguous sequence of dwords.

Bit String: A set of contiguous bits. In the 486 microprocessor bit strings can be up to 4 gigabits long.

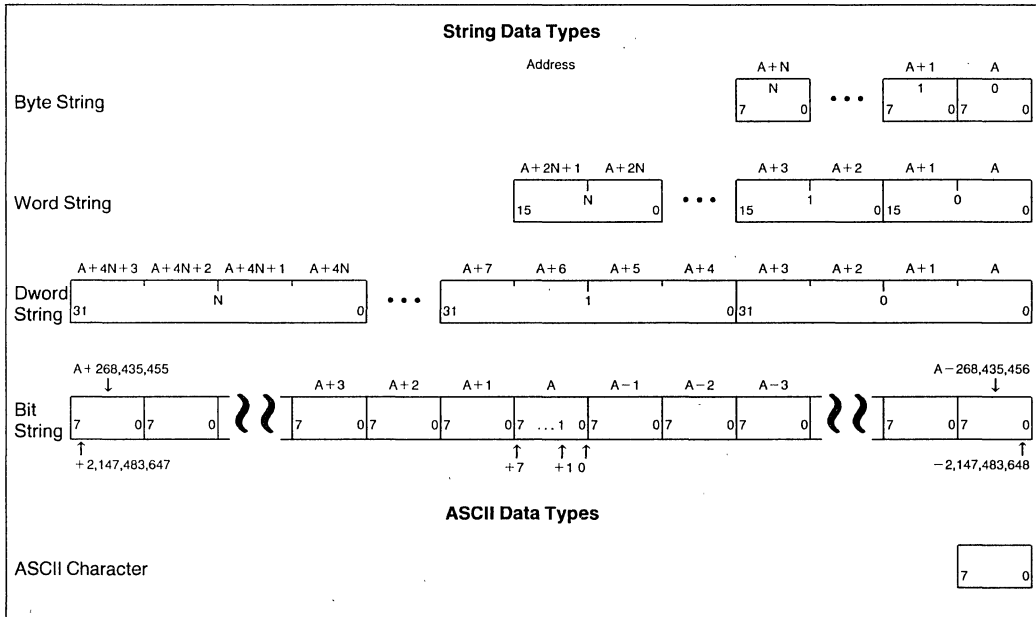
2.6.1.6 ASCII Data Types

The 486 microprocessor supports ASCII (American Standard Code for Information Interchange) strings and can perform arithmetic operations (such as addition and division) on ASCII data. The CPU can only operate on ASCII data. Refer to Table 2.14.

Table 2.13. i486™ Microprocessor Data Types

Data Format	Supported by		Range	Precision	Least Significant Byte															
	Base Registers	FPU			7	0	7	0	7	0	7	0	7	0	7	0	7	0		
Byte	X		0–255	8 bits																
Word	X		0–64K	16 bits																
Dword	X		0–4G	32 bits																
8-Bit Integer	X		10^2	8 bits																
16-Bit Integer	X	X	10^4	16 bits																
32-Bit Integer	X	X	10^9	32 bits																
64-Bit Integer		X	10^{19}	64 bits																
8-Bit Unpacked BCD	X		0–9	1 Digit																
8-Bit Packed BCD	X		0–9	2 Digits																
80-Bit Packed BCD	X		$\pm 10^{\pm 18}$	18 Digits																
Single Precision Real	X		$\pm 10^{\pm 38}$	24 Bits																
Double Precision Real	X		$\pm 10^{\pm 308}$	53 Bits																
Extended Precision Real	X		$\pm 10^{\pm 4932}$	64 Bits																

Table 2.14. String and ASCII Data Types



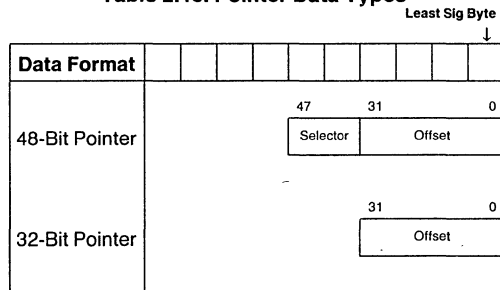
2.6.1.7 Pointer Data Types

A pointer data type contains a value that gives the address of a piece of data. The 486 microprocessor supports two types of pointers. Refer to Table 2.15.

48-bit Pointer: 16-bit selector and 32-bit offset

32-bit Pointer: 32-bit offset

Table 2.15. Pointer Data Types



2.6.2 LITTLE ENDIAN vs BIG ENDIAN DATA FORMATS

The 486 microprocessor, as well as all other members of the 86 architecture use the "little-endian" method for storing data types that are larger than one byte. Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address and the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address and the high order byte at the highest address. The address of a word or dword data item is the byte address of the low-order byte.

Figure 2.18 illustrates the differences between the big-endian and little-endian formats for dwords. The 32 bits of data are shown with the low order bit numbered bit 0 and the high order bit numbered 32. Big-endian data is stored with the high-order bits at the lowest addressed byte. Little-endian data is stored with the high-order bits in the highest addressed byte.

The 486 microprocessor has two instructions which can convert 16- or 32-bit data between the two byte orderings. BSWAP (byte swap) handles four byte values and XCHG (exchange) handles two byte values.

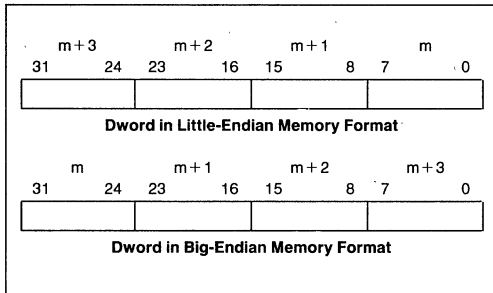


Figure 2.18. Big vs Little Endian Memory Format

2.7 Interrupts

2.7.1 INTERRUPTS AND EXCEPTIONS

Interrupts and exceptions alter the normal program flow, in order to handle external events, to report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT N instruction, the processor treats software interrupts as exceptions.

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately **after** the interrupted instruction. Sections 2.7.3 and 2.7.4 discuss the differences between Maskable and Non-Maskable interrupts.

Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is supported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. A fault would occur in a virtual memory system, when the processor referenced a page or a segment which was not present. The operating system would fetch the page or segment from disk, and then the 486 Microprocessor would restart the instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. User defined interrupts are examples of traps. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined. Aborts are used to report severe errors, such as a hardware error, or illegal values in system tables.

Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point at the instruction causing the exception and include any leading instruction prefixes. Table 2.16 summarizes the possible interrupts for the 486 Microprocessor and shows where the return address points.

The 486 Microprocessor has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode (see Section 3.1), the vectors are 4 byte quantities, a Code Segment plus a 16-bit offset; in Protected Mode, the interrupt vectors are 8 byte quantities, which are put in an Interrupt Descriptor Table (see Section 4.3.3.4). Of the 256 possible interrupts, 32 are reserved for use by Intel, the remaining 224 are free to be used by the system designer.

2.7.2 INTERRUPT PROCESSING

When an interrupt occurs the following actions happen. First, the current program address and the Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the 486 Microprocessor which identifies the

appropriate entry in the interrupt table. The table contains the starting address of the interrupt service routine. Then, the user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the 486 Microprocessor in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

2.7.3 MASKABLE INTERRUPT

Maskable interrupts are the most common way used by the 486 Microprocessor to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled high and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions, (REPEAT String instructions, have an "interrupt window", between memory moves, which allows interrupts during long string moves). When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt, (one of 224 user defined interrupts). The exact nature of the interrupt sequence is discussed in Section 7.2.10.

Table 2.16. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	YES	FAULT
Debug Exception	1	Any Instruction	YES	TRAP*
NMI Interrupt	2	INT 2 or NMI	NO	NMI
One Byte Interrupt	3	INT	NO	TRAP
Interrupt on Overflow	4	INTO	NO	TRAP
Array Bounds Check	5	BOUND	YES	FAULT
Invalid OP-Code	6	Any Illegal Instruction	YES	FAULT
Device Not Available	7	ESC, WAIT	YES	FAULT
Double Fault	8	Any Instruction That Can Generate an Exception		ABORT
Intel Reserved	9			
Invalid TSS	10	JMP, CALL, IRET, INT	YES	FAULT
Segment Not Present	11	Segment Register Instructions	YES	FAULT
Stack Fault	12	Stack References	YES	FAULT
General Protection Fault	13	Any Memory Reference	YES	FAULT
Page Fault	14	Any Memory Access or Code Fetch	YES	FAULT
Intel Reserved	15			
Floating Point Error	16	Floating Point, WAIT	YES	FAULT
Alignment Check Interrupt	17	Unaligned Memory Access	YES	FAULT
Intel Reserved	18-32			
Two Byte Interrupt	0-255	INT n	NO	TRAP

*Some debug exceptions may report both traps on the previous instruction, and faults on the next instruction.

The IF bit in the EFLAG registers is reset when an interrupt is being serviced. This effectively disables servicing additional interrupts during an interrupt service routine. However, the IF may be set explicitly by the interrupt handler, to allow the nesting of interrupts. When an IRET instruction is executed the original state of the IF is restored.

2.7.4 NON-MASKABLE INTERRUPT

Non-maskable interrupts provide a method of servicing very high priority interrupts. A common example of the use of a non-maskable interrupt (NMI) would be to activate a power failure routine. When the NMI input is pulled high it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt, no interrupt acknowledgment sequence is performed for an NMI.

While executing the NMI servicing procedure, the 486 Microprocessor will not service further NMI requests until an interrupt return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The IF bit is cleared at the beginning of an NMI interrupt to inhibit further INTR interrupts.

2.7.5 SOFTWARE INTERRUPTS

A third type of interrupt/exception for the 486 Microprocessor is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table.

A special case of the two byte software interrupt INT n is the one byte INT 3, or breakpoint interrupt. By inserting this one byte instruction in a program, the user can set breakpoints in his program as a debugging tool.

A final type of software interrupt is the single step interrupt. It is discussed in Section 9.2.

2.7.6 INTERRUPT AND EXCEPTION PRIORITIES

Interrupts are externally-generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are **both** recognized at the **same** instruction boundary, the 486 Microprocessor invokes the NMI service routine first. If, after the NMI service routine has been invoked, maskable interrupts are still enabled, then the 486 Microprocessor will invoke the appropriate interrupt service routine.

Table 2.17a. i486™ Microprocessor Priority for Invoking Service Routines in Case of Simultaneous External Interrupts

1. NMI
2. INTR

Exceptions are internally-generated events. Exceptions are detected by the 486 Microprocessor if, in the course of executing an instruction, the 486 Microprocessor detects a problematic condition. The 486 Microprocessor then immediately invokes the appropriate exception service routine. The state of the 486 Microprocessor is such that the instruction causing the exception can be restarted. If the exception service routine has taken care of the problematic condition, the instruction will execute without causing the same exception.

It is possible for a single instruction to generate several exceptions (for example, transferring a single operand could generate two page faults if the operand location spans two "not present" pages). However, only one exception is generated upon each attempt to execute the instruction. Each exception service routine should correct its corresponding exception, and restart the instruction. In this manner, exceptions are serviced until the instruction executes successfully.

As the 486 Microprocessor executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2.17b. This cycle is repeated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

Table 2.17b. Sequence of Exception Checking

Consider the case of the 486 Microprocessor having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
3. Check for external NMI and INTR.
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Page Faults that prevented fetching the entire next instruction (exception 14).
6. Check for Faults decoding the next instruction (exception 6 if illegal opcode; exception 6 if in Real Mode or in Virtual 8086 Mode and attempting to execute an instruction for Protected Mode only (see Section 4.6.4); or exception 13 if instruction is longer than 15 bytes, or privilege violation in Protected Mode (i.e., not at IOPL or at CPL=0).
7. If WAIT opcode, check if TS=1 and MP=1 (exception 7 if both are 1).
8. If opcode for Floating Point Unit, check if EM=1 or TS=1 (exception 7 if either are 1).
9. If opcode for Floating Point Unit (FPU), check FPU error status (exception 16 if error status is asserted).
10. Check in the following order for each memory reference required by the instruction:
 - a. Check for Segmentation Faults that prevent transferring the entire memory quantity (exceptions 11, 12, 13).
 - b. Check for Page Faults that prevent transferring the entire memory quantity (exception 14).

NOTE:

The order stated supports the concept of the paging mechanism being “underneath” the segmentation mechanism. Therefore, for any given code or data reference in memory, segmentation exceptions are generated before paging exceptions are generated.

2.7.7 INSTRUCTION RESTART

The 486 Microprocessor fully supports restarting all instructions after faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10 in Table 2.17b), the 486 Microprocessor invokes the appropriate exception service routine. The 486 Microprocessor is in a state that permits restart of the instruction, for all cases but those in Table 2.17c. Note that all such cases are easily avoided by proper design of the operating system.

Table 2.17c. Conditions Preventing Instruction Restart

An instruction causes a task switch to a task whose Task State Segment is **partially** “not present”. (An entirely “not present” TSS is restartable.) Partially present TSS’s can be avoided either by keeping the TSS’s of such tasks present in memory, or by aligning TSS segments to reside entirely within a single 4K page (for TSS segments of 4 Kbytes or less).

NOTE:

These conditions are avoided by using the operating system designs mentioned in this table.

2.7.8 DOUBLE FAULT

A Double Fault (exception 8) results when the processor attempts to invoke an exception service routine for the segment exceptions (10, 11, 12 or 13), but in the process of doing so, detects an exception other than a Page Fault (exception 14).

A Double Fault (exception 8) will also be generated when the processor attempts to invoke the Page Fault (exception 14) service routine, and detects an exception other than a second Page Fault. In any functional system, the entire Page Fault service routine must remain “present” in memory.

When a Double Fault occurs, the 486 Microprocessor invokes the exception service routine for exception 8.

2.7.9 FLOATING POINT INTERRUPT VECTORS

Several interrupt vectors of the 486 microprocessor are used to report exceptional conditions while executing numeric programs in either real or protected mode. Table 2.18 shows these interrupts and their causes.

Table 2.18. Interrupt Vectors Used by FPU

Interrupt Number	Cause of Interrupt
7	A Floating Point instruction was encountered when EM or TS of the 486™ processor control register zero (CR0) was set. EM = 1 indicates that software emulation of the instruction is required. When TS is set, either a Floating Point or WAIT instruction causes interrupt 7. This indicates that the current FPU context may not belong to the current task.
13	The first word or doubleword of a numeric operand is not entirely within the limit of its segment. The return address pushed onto the stack of the exception handler points at the Floating Point instruction that caused the exception, including any prefixes. The FPU has not executed this instruction; the instruction pointer and data pointer register refer to a previous, correctly executed instruction.
16	The previous numerics instruction caused an unmasked exception. The address of the faulty instruction and the address of its operand are stored in the instruction pointer and data pointer registers. Only Floating Point and WAIT instructions can cause this interrupt. The 486™ processor return address pushed onto the stack of the exception handler points to a WAIT or Floating Point instruction (including prefixes). This instruction can be restarted after clearing the exception condition in the FPU. The FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE instructions cannot cause this interrupt.

3.0 REAL MODE ARCHITECTURE

3.1 Real Mode Introduction

When the processor is reset or powered up it is initialized in Real Mode. Real Mode has the same base architecture as the 8086, but allows access to the 32-bit register set of the 486 Microprocessor. The addressing mechanism, memory size, interrupt handling, are all identical to the Real Mode on the 80286.

All of the 486 Microprocessor instructions are available in Real Mode (except those instructions listed in Section 4.6.4). The default operand size in Real Mode is 16 bits, just like the 8086. In order to use the 32-bit registers and addressing modes, override prefixes must be used. In addition, the segment size on the 486 Microprocessor in Real Mode is 64 Kbytes so 32-bit effective addresses must have a value less the 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode Operation.

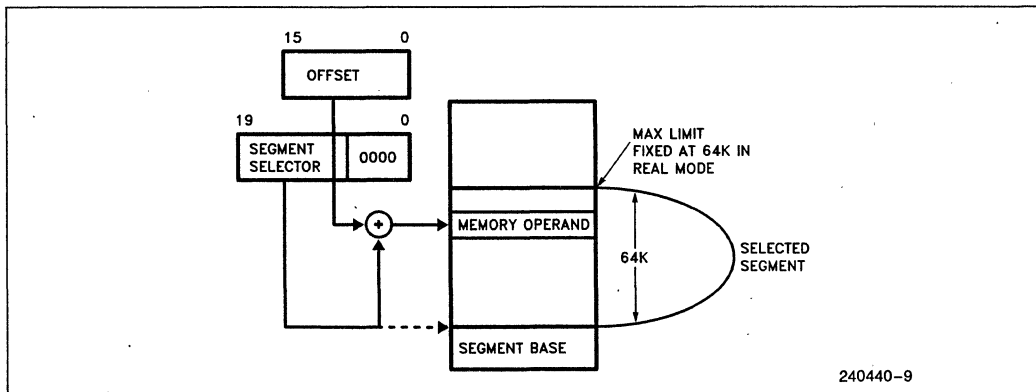


Figure 3.1. Real Address Mode Addressing

The LOCK prefix on the 486 Microprocessor, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the 486 Microprocessor in Protected Mode and Virtual 8086 Mode. Paging makes it impossible to guarantee that repeated string instructions can be LOCKed. The 486 Microprocessor can't require that all pages holding the string be physically present in memory. Hence, a Page Fault (exception 14) might have to be taken during the repeated string instruction. Therefore the LOCK prefix can't be supported during repeated string instructions.

These are the only instruction forms where the LOCK prefix is legal on the 486 Microprocessor:

Opcode	Operands (Dest, Source)
BIT Test and SET/RESET/COMPLEMENT	Mem, Reg/immed
XCHG	Reg, Mem
XCHG	Mem, Reg
ADD, OR, ADC, SBB, AND, SUB, XOR	Mem, Reg/immed
NOT, NEG, INC, DEC	Mem

An exception 6 will be generated if a LOCK prefix is placed before any instruction form or opcode not listed above. The LOCK prefix allows indivisible read/modify/write operations on memory operands using the instructions above. For example, even the ADD Reg, Mem is not LOCKable, because the Mem operand is not the destination (and therefore no memory read/modify/operation is being performed).

Since, on the 486 Microprocessor, repeated string instructions are not LOCKable, it is not possible to LOCK the bus for a long period of time. Therefore, the LOCK prefix is not IOPL-sensitive on the 486 Microprocessor. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed above.

3.2 Memory Addressing

In Real Mode the maximum memory size is limited to 1 megabyte. Thus, only address lines A2–A19 are active. (Exception, after RESET address lines A20–A31 are high during CS-relative memory cycles until an intersegment jump or call is executed (see Section 6.5)).

Since paging is not allowed in Real Mode the linear addresses are the same as physical addresses. Physical addresses are formed in Real Mode by adding the contents of the appropriate segment register which is shifted left by four bits to an effective

address. This addition results in a physical address from 00000000H to 0010FFEFH. This is compatible with 80286 Real Mode. Since segment registers are shifted left by 4 bits, Real Mode segments always start on 16 byte boundaries.

All segments in Real Mode are exactly 64 Kbytes long, and may be read, written, or executed. The 486 Microprocessor will generate an exception 13 if a data operand or instruction fetch occurs past the end of a segment (i.e., if an operand has an offset greater than FFFFH, for example a word with a low byte at FFFFH and the high byte at 0000H).

Segments may be overlapped in Real Mode. Thus, if a particular segment does not use all 64 Kbytes another segment can be overlaid on top of the unused portion of the previous segment. This allows the programmer to minimize the amount of physical memory needed for a program.

3.3 Reserved Locations

There are two fixed areas in memory which are reserved in Real address mode: system initialization area and the interrupt table area. Locations 00000H through 003FFH are reserved for interrupt vectors. Each one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations FFFFFFF0H through FFFFFFFFH are reserved for system initialization.

3.4 Interrupts

Many of the exceptions shown in Table 2.16 and discussed in Section 2.7 are not applicable to Real Mode operation, in particular exceptions 10, 11, 14, will not happen in Real Mode. Other exceptions have slightly different meanings in Real Mode; Table 3.1 identifies these exceptions.

3.5 Shutdown and Halt

The HLT instruction stops program execution and prevents the processor from using the local bus until restarted. Either NMI, INTR with interrupts enabled (IF = 1), or RESET will force the 486 Microprocessor out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

Shutdown will occur when a severe error is detected that prevents further processing. In Real Mode, shutdown can occur under two conditions:

An interrupt or an exception occur (exceptions 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table (i.e., there is not an interrupt handler for the interrupt).

Table 3.1. Exceptions with Different Meanings in Real Mode (see Table 2.16)

Function	Interrupt Number	Related Instructions	Return Address Location
Interrupt table limit too small	8	INT Vector is not within table limit	Before Instruction
CS, DS, ES, FS, GS Segment overrun exception	13	Word memory reference beyond offset = FFFFH. An attempt to execute past the end of CS segment.	Before Instruction
SS Segment overrun exception	12	Stack Reference beyond offset = FFFFH	Before Instruction

A CALL, INT or PUSH instruction attempts to wrap around the stack segment when SP is not even (i.e., pushing a value on the stack when SP = 0001 resulting in a stack segment greater than FFFFH).

An NMI input can bring the processor out of shutdown if the Interrupt Descriptor Table limit is large enough to contain the NMI interrupt vector (at least 0017H) and the stack has enough room to contain the vector and flag information (i.e., SP is greater than 0005H). Otherwise shutdown can only be exited via the RESET input.

4.0 PROTECTED MODE ARCHITECTURE

4.1 Introduction

The complete capabilities of the 486 Microprocessor are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to four gigabytes (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 terabytes or 2^{46} bytes). In addition Protected Mode allows the 486 Microprocessor to run all of the existing 8086, 80286 and 386 microprocessor software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions especially optimized for supporting multitasking operating systems. The base architecture of the 486 Microprocessor remains the same, the registers, instructions, and addressing modes described in the previous sections are re-

tained. The main difference between Protected Mode, and Real Mode from a programmer's view is the increased address space, and a different addressing mechanism.

4.2 Addressing Mechanism

Like Real Mode, Protected Mode uses two components to form the logical address, a 16-bit selector is used to determine the linear base address of a segment, the base address is added to a 32-bit effective address to form a 32-bit linear address. The linear address is then either used as the 32-bit physical address, or if paging is enabled the paging mechanism maps the 32-bit linear address into a 32-bit physical address.

The difference between the two modes lies in calculating the base address. In Protected Mode the selector is used to specify an index into an operating system defined table (see Figure 4.1). The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base address obtained from the table to the offset.

Paging provides an additional memory management mechanism which operates only in Protected Mode. Paging provides a means of managing the very large segments of the 486 Microprocessor. As such, paging operates beneath segmentation. The paging mechanism translates the protected linear address which comes from the segmentation unit into a physical address. Figure 4.2 shows the complete 486 Microprocessor addressing mechanism with paging enabled.

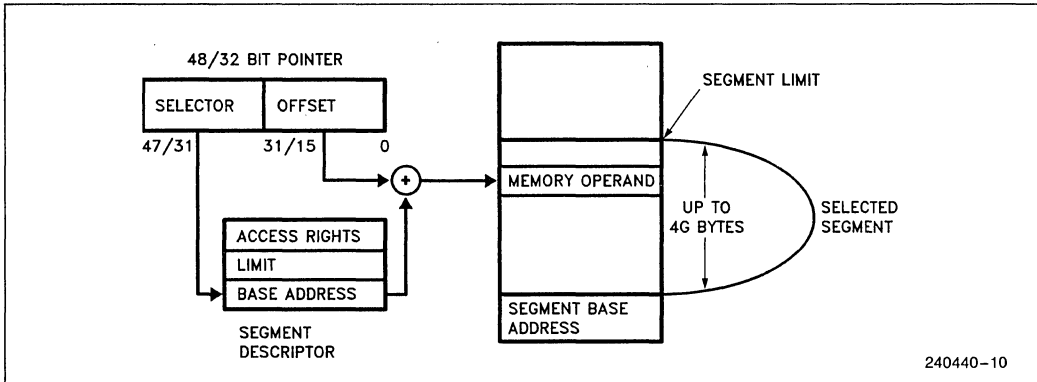


Figure 4.1. Protected Mode Addressing

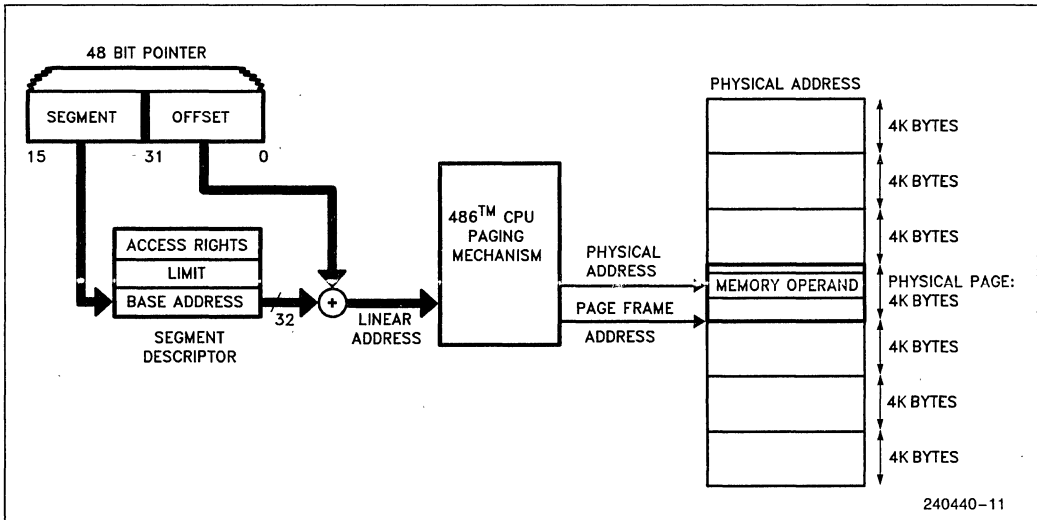


Figure 4.2. Paging and Segmentation

4.3 Segmentation

4.3.1 SEGMENTATION INTRODUCTION

Segmentation is one method of memory management. Segmentation provides the basis for protection. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about a segment is stored in an 8 byte data structure called a descriptor. All of the descriptors in a system are contained in tables recognized by hardware.

4.3.2 TERMINOLOGY

The following terms are used throughout the discussion of descriptors, privilege levels and protection:

PL: Privilege Level—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged. More privileged levels are numerically smaller than less privileged levels.

RPL: Requestor Privilege Level—The privilege level of the original supplier of the selector. RPL is determined by the **least two** significant bits of a selector.

DPL: Descriptor Privilege Level—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor) (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.

CPL: Current Privilege Level—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.

EPL: Effective Privilege Level—The effective privilege level is the least privileged of the RPL and DPL. Since smaller privilege level values indicate greater privilege, EPL is the numerical maximum of RPL and DPL.

Task: One instance of the execution of a program. Tasks are also referred to as processes.

4.3.3 DESCRIPTOR TABLES

4.3.3.1 Descriptor Tables Introduction

The descriptor tables define all of the segments which are used in an 486 Microprocessor system. There are three types of tables on the 486 Microprocessor which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays. They can range in size between 8 bytes and 64 Kbytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address, and the 16-bit limit of each table.

Each of the tables has a register associated with it, the GDTR, LDTR, and the IDTR (see Figure 4.3). The LGDT, LLDT, and LIDT instructions, load the base and limit of the Global, Local, and Interrupt Descriptor Tables, respectively, into the appropriate register. The SGDT, SLDT, and SIDT store the base and limit values. These tables are manipulated by the operating system. Therefore, the load descriptor table instructions are privileged instructions.

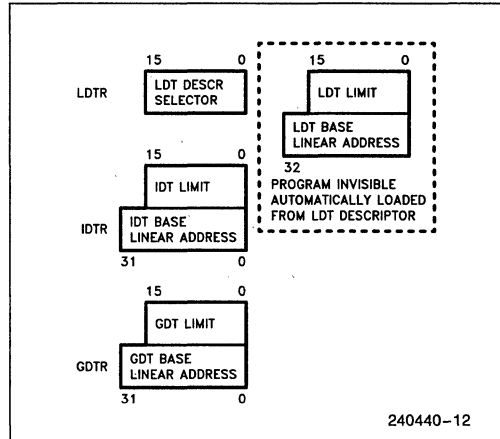


Figure 4.3. Descriptor Table Registers

4.3.3.2 Global Descriptor Table

The Global Descriptor Table (GDT) contains descriptors which are possibly available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for descriptors which are used for servicing interrupts (i.e., interrupt and trap descriptors). Every 486 Microprocessor system contains a GDT. Generally the GDT contains code and data segments used by the operating systems and task state segments, and descriptors for the LDTs in a system.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

4.3.3.3 Local Descriptor Table

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This pro-

vides both isolation and protection for a task's segments, while still allowing global data to be shared among tasks.

Unlike the 6 byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT.

4.3.3.4 Interrupt Descriptor Table

The third table needed for 486 Microprocessor systems is the Interrupt Descriptor Table. (See Figure 4.4.) The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced via INT instructions, external interrupt vectors, and exceptions. (See Section 2.7 **Interrupts**).

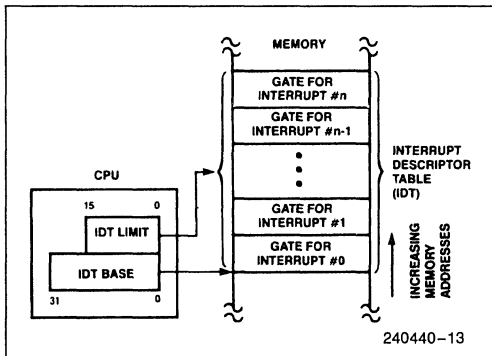


Figure 4.4. Interrupt Descriptor Table Register Use

4.3.4 DESCRIPTORS

4.3.4.1 Descriptor Attribute Bits

The object to which the segment selector points to is called a descriptor. Descriptors are eight byte quantities which contain attributes about a given region of linear address space (i.e., a segment). These attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or 32-bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 4.5 shows the general format of a descriptor. All segments on the 486 Microprocessor have three attribute fields in common: the **P** bit, the **DPL** bit, and the **S** bit. The Present **P** bit is 1 if the segment is loaded in physical memory, if P=0 then any attempt to access this segment causes a not present exception (exception 11). The Descriptor Privilege Level **DPL** is a two-bit field which specifies the protection level 0-3 associated with a segment.

The 486 Microprocessor has two main categories of segments: system segments and non-system segments (for code and data). The segment **S** bit in the segment descriptor determines if a given segment is a system segment or a code or data segment. If the **S** bit is 1 then the segment is either a code or data segment, if it is 0 then the segment is a system segment.

4.3.4.2 i486™ CPU Code, Data Descriptors (S=1)

Figure 4.6 shows the general format of a code and data descriptor and Table 4.1 illustrates how the bits in the Access Rights Byte are interpreted.

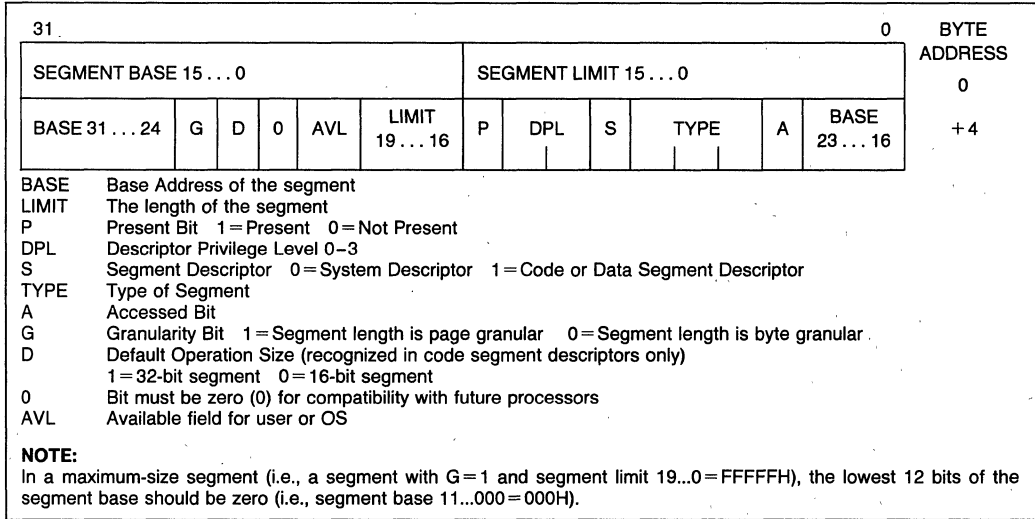


Figure 4.5. Segment Descriptors

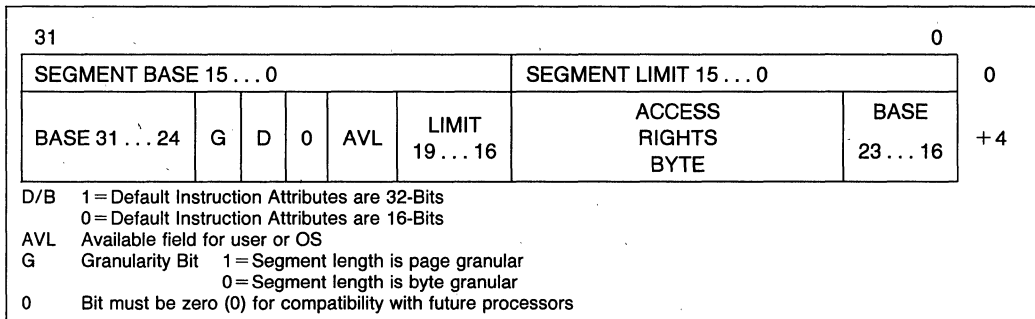


Figure 4.6. Segment Descriptors

Table 4.1. Access Rights Byte Definition for Code and Data Descriptors

Bit Position	Name	Function
7	Present (P)	P = 1 Segment is mapped into physical memory. P = 0 No mapping to physical memory exists, base and limit are not used.
6–5	Descriptor Privilege Level (DPL)	Segment privilege attribute used in privilege tests.
4	Segment Descriptor (S)	S = 1 Code or Data (includes stacks) segment descriptor. S = 0 System Segment Descriptor or Gate Descriptor.
3	Executable (E)	E = 0 Descriptor type is data segment:
2	Expansion Direction (ED)	ED = 0 Expand up segment, offsets must be ≤ limit.
1	Writeable (W)	ED = 1 Expand down segment, offsets must be > limit. W = 0 Data segment may not be written into. W = 1 Data segment may be written into.
Type Field Definition	3	Executable (E)
	2	Conforming (C)
	1	Readable (R)
0	Accessed (A)	A = 0 Segment has not been accessed. A = 1 Segment selector has been loaded into segment register or used by selector test instructions.

Code and data segments have several descriptor fields in common. The accessed **A** bit is set whenever the processor accesses a descriptor. The **A** bit is used by operating systems to keep usage statistics on a given segment. The **G** bit, or granularity bit, specifies if a segment length is byte-granular or page-granular. 486 Microprocessor segments can be one megabyte long with byte granularity ($G=0$) or four gigabytes with page granularity ($G=1$), (i.e., 2²⁰ pages each page is 4 Kbytes in length). The granularity is totally unrelated to paging. A 486 Microprocessor system can consist of segments with byte granularity, and page granularity, whether or not paging is enabled.

The executable **E** bit tells if a segment is a code or data segment. A code segment ($E=1, S=1$) may be execute-only or execute/read as determined by the Read **R** bit. Code segments are execute only if $R=0$, and execute/read if $R=1$. Code segments may never be written into.

NOTE:

Code segments may be modified via aliases. Aliases are writeable data segments which occupy the same range of linear address space as the code segment.

The **D** bit indicates the default length for operands and effective addresses. If $D=1$ then 32-bit operands and 32-bit addressing modes are assumed. If $D=0$ then 16-bit operands and 16-bit addressing modes are assumed. Therefore all existing 80286 code segments will execute on the 486 Microprocessor assuming the **D** bit is set 0.

Another attribute of code segments is determined by the conforming **C** bit. Conforming segments, $C=1$, can be executed and shared by programs at different privilege levels. (See Section 4.4 **Protection**.)

Segments identified as data segments ($E=0, S=1$) are used for two types of 486 Microprocessor segments: stack and data segments. The expansion direction (**ED**) bit specifies if a segment expands downward (stack) or upward (data). If a segment is a stack segment all offsets must be greater than the segment limit. On a data segment all offsets must be less than or equal to the limit. In other words, stack segments start at the base linear address plus the maximum segment limit and grow down to the base linear address plus the limit. On the other hand, data segments start at the base linear address and expand to the base linear address plus limit.

The write **W** bit controls the ability to write into a segment. Data segments are read-only if $W=0$. The stack segment must have $W=1$.

The **B** bit controls the size of the stack pointer register. If $B=1$, then PUSHes, POPs, and CALLs all use the 32-bit ESP register for stack references and assume an upper limit of FFFFFFFFH. If $B=0$, stack instructions all use the 16-bit SP register and assume an upper limit of FFFFH.

4.3.4.3 System Descriptor Formats

System segments describe information about operating system tables, tasks, and gates. Figure 4.7 shows the general format of system segment descriptors, and the various types of system segments. 486 Microprocessor system descriptors contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.

4.3.4.4 LDT Descriptors (S=0, TYPE=2)

LDT descriptors ($S=0$, $TYPE=2$) contain information about Local Descriptor Tables. LDTs contain a table of segment descriptors, unique to a particular task. Since the instruction to load the LDTR is only available at privilege level 0, the DPL field is ignored. LDT descriptors are only allowed in the Global Descriptor Table (GDT).

4.3.4.5 TSS Descriptors (S=0, TYPE=1, 3, 9, B)

A Task State Segment (TSS) descriptor contains information about the location, size, and privilege level of a Task State Segment (TSS). A TSS in turn is a special fixed format segment which contains all the state information for a task and a linkage field to permit nesting tasks. The TYPE field is used to indicate whether the task is currently BUSY (i.e., on a chain of active tasks) or the TSS is available. The TYPE field also indicates if the segment contains a 80286 or a 486 Microprocessor TSS. The Task Register (TR) contains the selector which points to the current Task State Segment.

4.3.4.6 Gate Descriptors (S=0, TYPE=4-7, C, F)

Gates are used to control access to entry points within the target code segment. The various types of gate descriptors are **call** gates, **task** gates, **interrupt** gates, and **trap** gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the processor to automatically perform protection checks. It also allows system designers to control entry points to the operating system. Call gates are used to change privilege levels (see Section 4.4 **Protection**), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines.

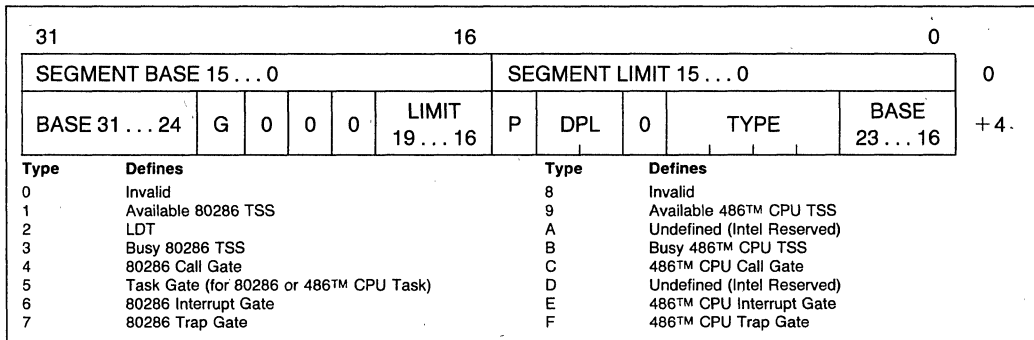


Figure 4.7. System Segment Descriptors

Figure 4.8 shows the format of the four types of gate descriptors. Call gates are primarily used to transfer program control to a more privileged level. The call gate descriptor consists of three fields: the access byte, a long pointer (selector and offset) which points to the start of a routine and a word count which specifies how many parameters are to be copied from the caller's stack to the stack of the called routine. The word count field is only used by call gates when there is a change in the privilege level, other types of gates ignore the word count field.

Interrupt and trap gates use the destination selector and destination offset fields of the gate descriptor as a pointer to the start of the interrupt or trap handler routines. The difference between interrupt gates and trap gates is that the interrupt gate disables interrupts (resets the IF bit) while the trap gate does not.

Task gates are used to switch tasks. Task gates may only refer to a task state segment (see Section 4.4.6 **Task Switching**) therefore only the destination selector portion of a task gate descriptor is used, and the destination offset is ignored.

Exception 13 is generated when a destination selector does not refer to a correct descriptor type, i.e., a code segment for an interrupt, trap or call gate, a TSS for a task gate.

The access byte format is the same for all gate descriptors. P=1 indicates that the gate contents are valid. P=0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (see Section 4.4 **Protection**). The S field, bit 4 of the access rights byte, must be 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 4.8.

4.3.4.7 Differences Between i486™ Microprocessor and 80286 Descriptors

In order to provide operating system compatibility between the 80286 and 486 Microprocessor, the 486 Microprocessor supports all of the 80286 segment descriptors. Figure 4.9 shows the general format of an 80286 system segment descriptor. The only differences between 80286 and 486 Microprocessor descriptor formats are that the values of the type fields, and the limit and base address fields have been expanded for the 486 Microprocessor. The 80286 system segment descriptors contained a 24-bit base address and 16-bit limit, while the 486 Microprocessor system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit.

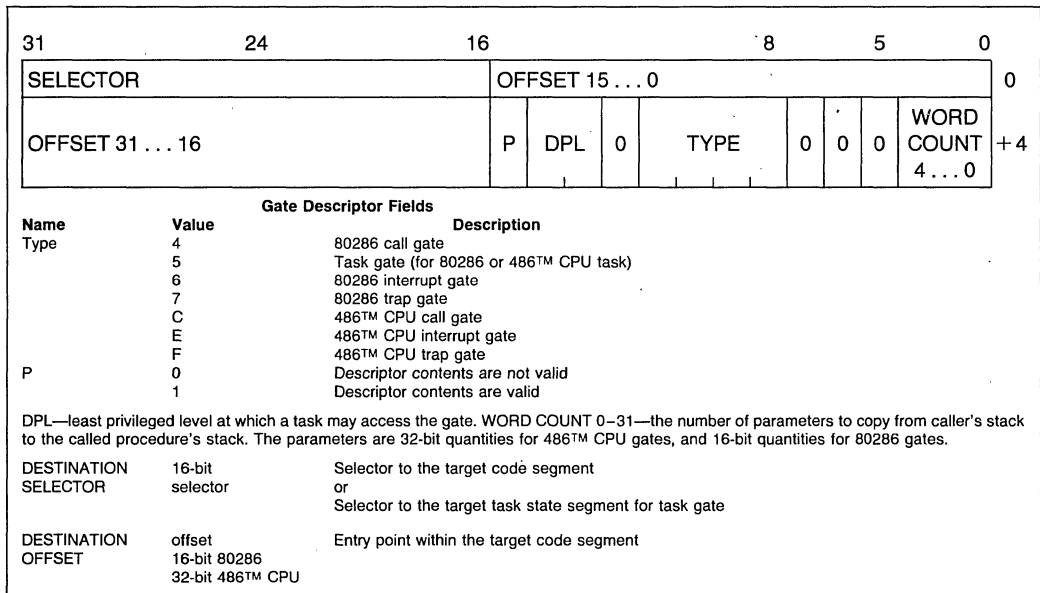


Figure 4.8. Gate Descriptor Formats

By supporting 80286 system segments the 486 Microprocessor is able to execute 80286 application programs on a 486 Microprocessor operating system. This is possible because the processor automatically understands which descriptors are 80286-style descriptors and which descriptors are 486 Microprocessor-style descriptors. In particular, if the upper word of a descriptor is zero, then that descriptor is a 80286-style descriptor.

The only other differences between 80286-style descriptors and 486 Microprocessor descriptors is the interpretation of the word count field of call gates and the B bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for 486 Microprocessor call gates. The B bit controls the size of PUSHes when using a call gate; if B = 0 PUSHes are 16 bits, if B = 1 PUSHes are 32 bits.

4.3.4.8 Selector Fields

A selector in Protected Mode has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor

Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 4.10. The TI bits select one of two memory-based tables of descriptors (the Global Descriptor Table or the Local Descriptor Table). The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

4.3.4.9 Segment Descriptor Cache

In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Since descriptor caches only change when a segment register is changed, programs which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

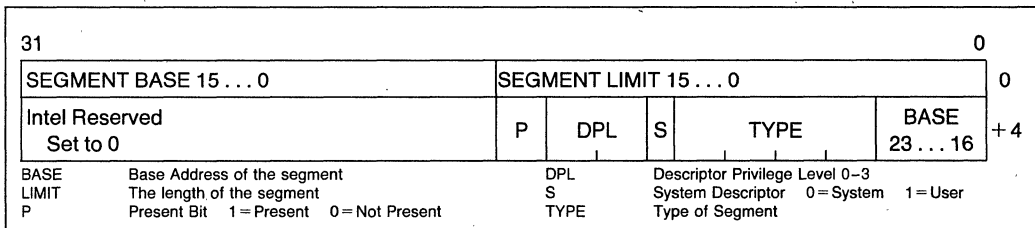


Figure 4.9. 80286 Code and Data Segment Descriptors

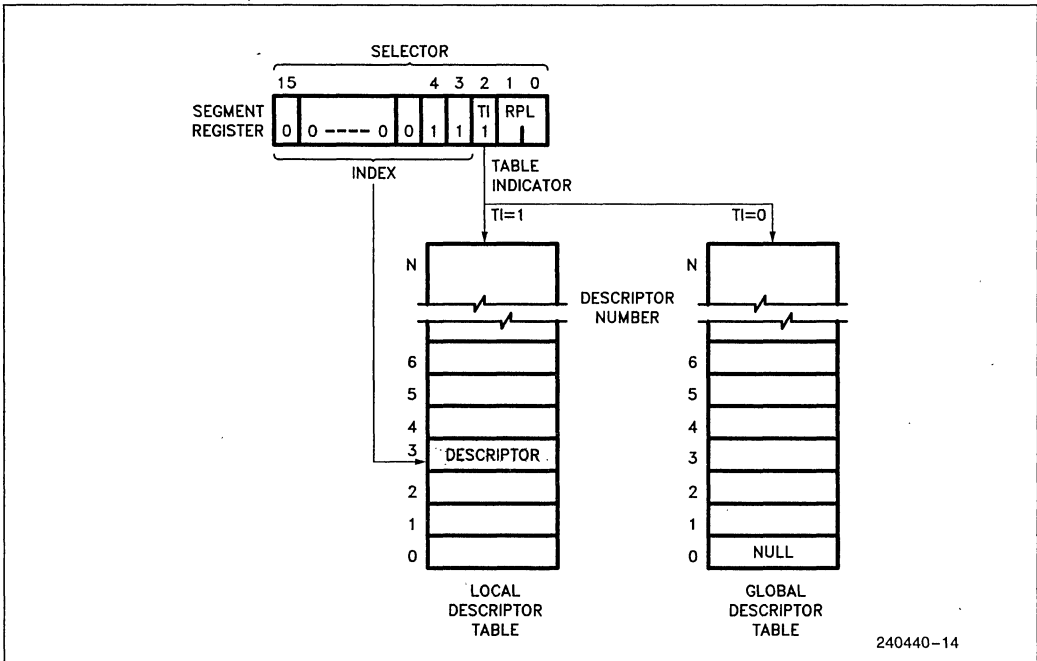
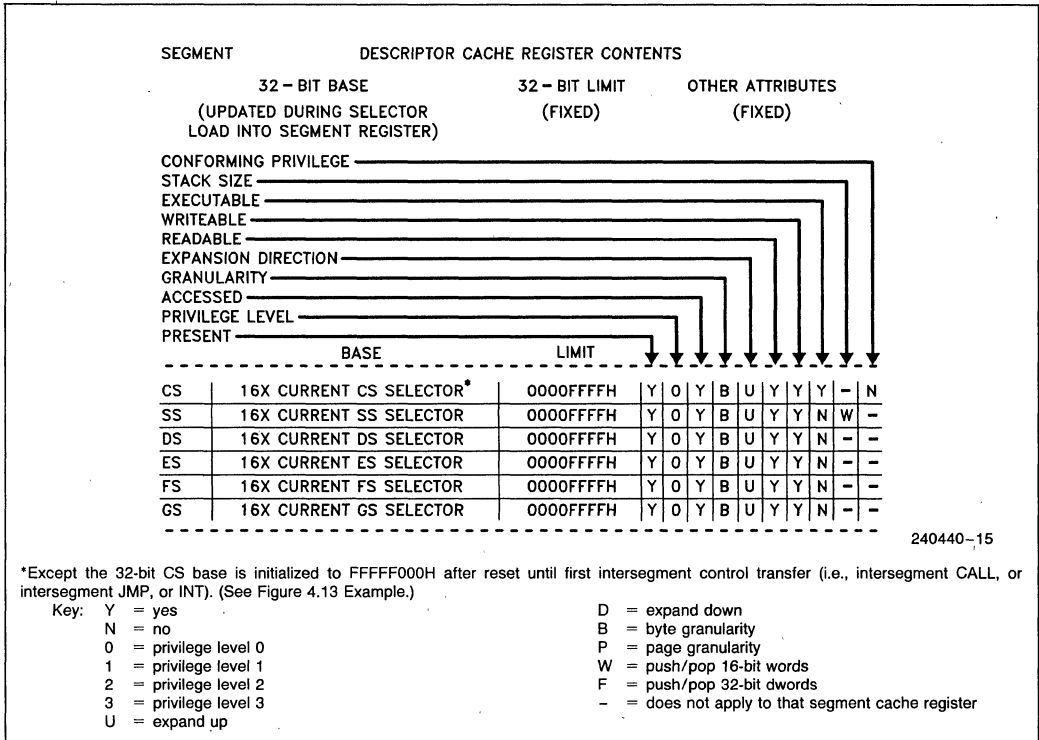


Figure 4.10. Example Descriptor Selection

4.3.4.10 Segment Descriptor Register Settings

The contents of the segment descriptor cache vary depending on the mode the 486 Microprocessor is operating in. When operating in Real Address Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.11. For compatibility with the 8086 archi-

ture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. In Real Address Mode, the internal "privilege level" is always fixed to the highest level, level 0, so I/O and other privileged opcodes may be executed.



**Figure 4.11. Segment Descriptor Caches for Real Address Mode
(Segment Limit and Attributes are Fixed)**

When operating in Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.12. In Protected Mode, each of these fields are defined

according to the contents of the segment descriptor indexed by the selector value loaded into the segment register.

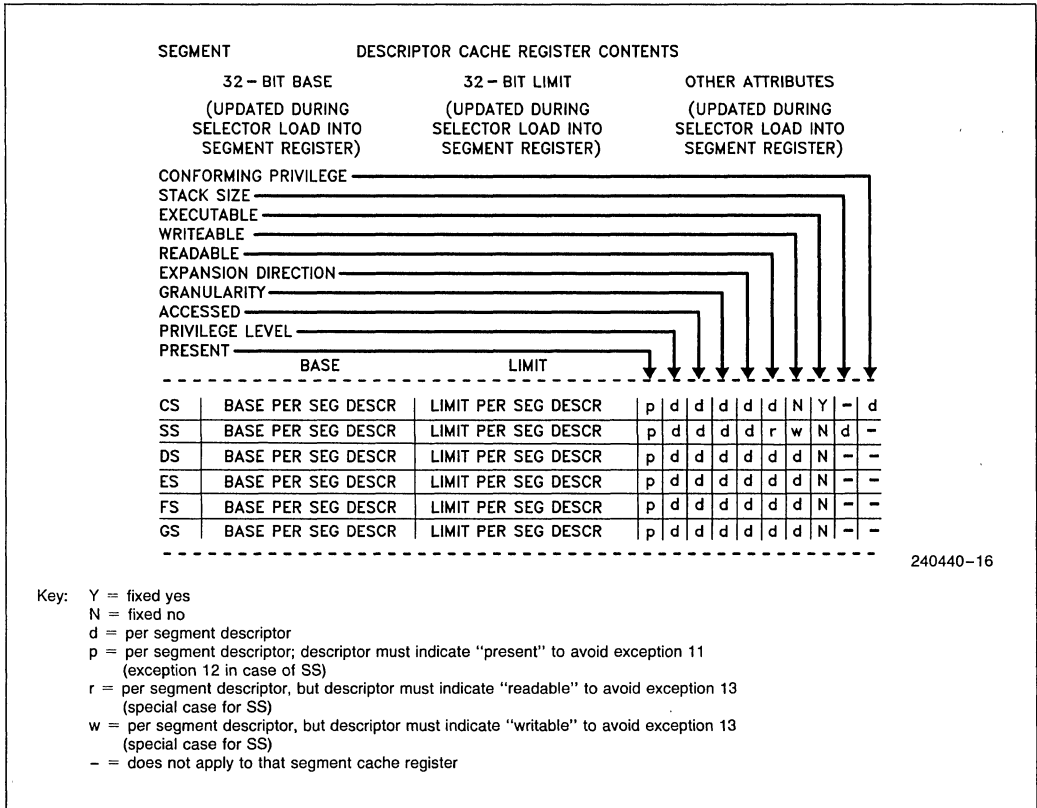


Figure 4.12. Segment Descriptor Caches for Protected Mode (Loaded per Descriptor)

When operating in a Virtual 8086 Mode within the Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.13. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at

0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. The virtual program executes at lowest privilege level, level 3, to allow trapping of all IOPL-sensitive instructions and level-0-only instructions.

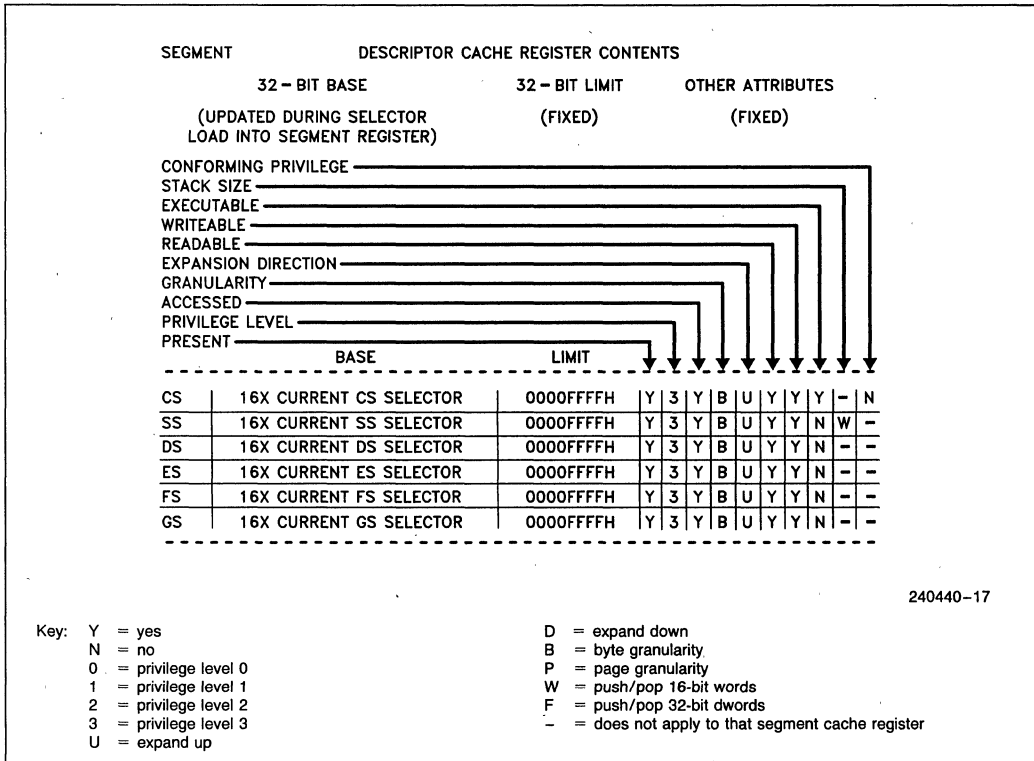


Figure 4.13. Segment Descriptor Caches for Virtual 8086 Mode within Protected Mode (Segment Limit and Attributes are Fixed)

4.4 Protection

4.4.1 PROTECTION CONCEPTS

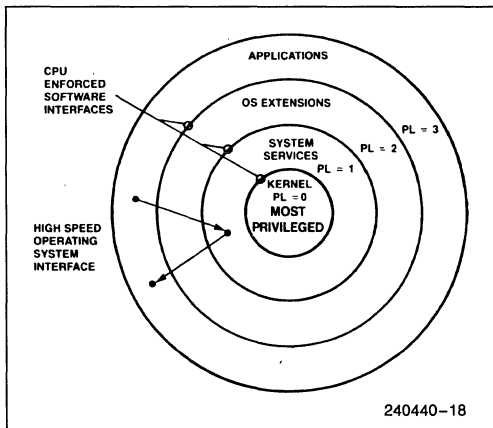


Figure 4.14. Four-Level Hierarchical Protection

The 486 Microprocessor has four levels of protection which are optimized to support the needs of a multi-tasking operating system to isolate and protect user programs from each other and the operating system. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. Unlike traditional microprocessor-based systems where this protection is achieved only through the use of complex external hardware and software the 486 Microprocessor provides the protection as part of its integrated Memory Management Unit. The 486 Microprocessor offers an additional type of protection on a page basis, when paging is enabled (See Section 4.5.3 **Page Level Protection**).

The four-level hierarchical privilege system is illustrated in Figure 4-14. It is an extension of the user/supervisor privilege mode commonly used by mini-computers and, in fact, the user/supervisor mode is fully supported by the 486 Microprocessor paging

mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged or trusted level.

4.4.2 RULES OF PRIVILEGE

The 486 Microprocessor controls access to both data and procedures between levels of a task, according to the following rules.

- Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p**.
- A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p**.

4.4.3 PRIVILEGE LEVELS

4.4.3.1 Task Privilege

At any point in time, a task on the 486 Microprocessor always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies the task's privilege level. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. (See Section 4.4.4 **Privilege Level Transfers**) Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

4.4.3.2 Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The RPL is the two least significant bits of the selector. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (i.e. numerically larger) level of a task's CPL and a selector's RPL. Thus, if selector's RPL = 0 then the CPL always specifies the privilege level for making an access using the selector. On the other hand if RPL = 3 then a selector can only access segments at level

3 regardless of the task's CPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

4.4.3.3 I/O Privilege and I/O Permission Bitmap

The I/O privilege level (IOPL, a 2-bit field in the EFLAG register) defines the least privileged level at which I/O instructions can be unconditionally performed. I/O instructions can be unconditionally performed when $CPL \leq IOPL$. (The I/O instructions are IN, OUT, INS, OUTS, REP INS, and REP OUTS.) When $CPL > IOPL$, and the current task is associated with a 286 TSS, attempted I/O instructions cause an exception 13 fault. When $CPL > IOPL$, and the current task is associated with a 486 Microprocessor TSS, the I/O Permission Bitmap (part of a 486 Microprocessor TSS) is consulted on whether I/O to the port is allowed, or an exception 13 fault is to be generated instead. For diagrams of the I/O Permission Bitmap, refer to Figures 4.15a and 4.15b. For further information on how the I/O Permission Bitmap is used in Protected Mode or in Virtual 8086 Mode, refer to Section 4.6.4 Protection and I/O Permission Bitmap.

The I/O privilege level (IOPL) also affects whether several other instructions can be executed or cause an exception 13 fault instead. These instructions are called "IOPL-sensitive" instructions and they are CLI and STI. (Note that the LOCK prefix is *not* IOPL-sensitive on the 486 Microprocessor.)

The IOPL also affects whether the IF (interrupts enable flag) bit can be changed by loading a value into the EFLAGS register. When $CPL \leq IOPL$, then the IF bit can be changed by loading a new value into the EFLAGS register. When $CPL > IOPL$, the IF bit cannot be changed by a new value POP'ed into (or otherwise loaded into) the EFLAGS register; the IF bit merely remains unchanged and no exception is generated.

Table 4.2. Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

4.4.3.4 Privilege Validation

The 486 Microprocessor provides several instructions to speed pointer testing and help maintain system integrity by verifying that the selector value refers to an appropriate segment. Table 4.2 summarizes the selector validation procedures available for the 486 Microprocessor.

This pointer verification prevents the common problem of an application at PL = 3 calling a operating systems routine at PL = 0 and passing the operating system routine a "bad" pointer which corrupts a data structure belonging to the operating system. If the operating system routine uses the ARPL instruc-

tion to ensure that the RPL of the selector has no greater privilege than that of the caller, then this problem can be avoided.

4.4.3.5 Descriptor Access

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL as described above.

Any time an instruction loads data segment registers (DS, ES, FS, GS) the 486 Microprocessor makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segments or readable code segments. The data access rules are specified in Section 4.4.2 **Rules of Privilege**. The only exception to those rules is readable conforming code segments which can be accessed at any privilege level.

Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL an exception 13 (general protection fault) is generated.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data segments. The DPL and RPL must equal the CPL. All other descriptor types or a privilege level violation will cause exception 13. A stack not present fault causes exception 12. Note that an exception 11 is used for a not-present code or data segment.

4.4.4 PRIVILEGE LEVEL TRANSFERS

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 4.3. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only via control transfers, by using gates, task switches, and interrupt or trap gates.

Table 4.3. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag register) = 0

**NT (Nested Task bit of flag register) = 1

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13 (e.g. JMP through a call gate, or IRET from a normal subroutine call).

In order to provide further system security, all control transfers are also subject to the privilege rules.

The privilege rules require that:

- Privilege level transitions can only occur via gates.
- JMPs can be made to a non-conforming code segment with the same privilege or to a conforming code segment with greater or equal privilege.
- CALLs can be made to a non-conforming code segment with the same privilege or via a gate to a more privileged level.
- Interrupts handled within the task obey the same privilege rules as CALLs.
- Conforming Code segments are accessible by privilege levels which are the same or less privileged than the conforming-code segment's DPL.
- Both the requested privilege level (RPL) in the selector pointing to the gate and the task's CPL must be of equal or greater privilege than the gate's DPL.
- The code segment selected in the gate must be the same or more privileged than the task's CPL.

- Return instructions that do not switch tasks can only return control to a code segment with same or less privilege.
- Task switches can be performed by a CALL, JMP, or INT which references either a task gate or task state segment who's DPL is less privileged or the same privilege as the old task's CPL.

Any control transfer that changes CPL within a task causes a change of stacks as a result of the privilege level change. The initial values of SS:ESP for privilege levels 0, 1, and 2 are retained in the task state segment (see Section 4.4.6 **Task Switching**). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and ESP registers and the previous stack pointer is pushed onto the new stack.

When RETURNing to the original privilege level, use of the lower-privileged stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words (as specified in the gate's word count field) are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

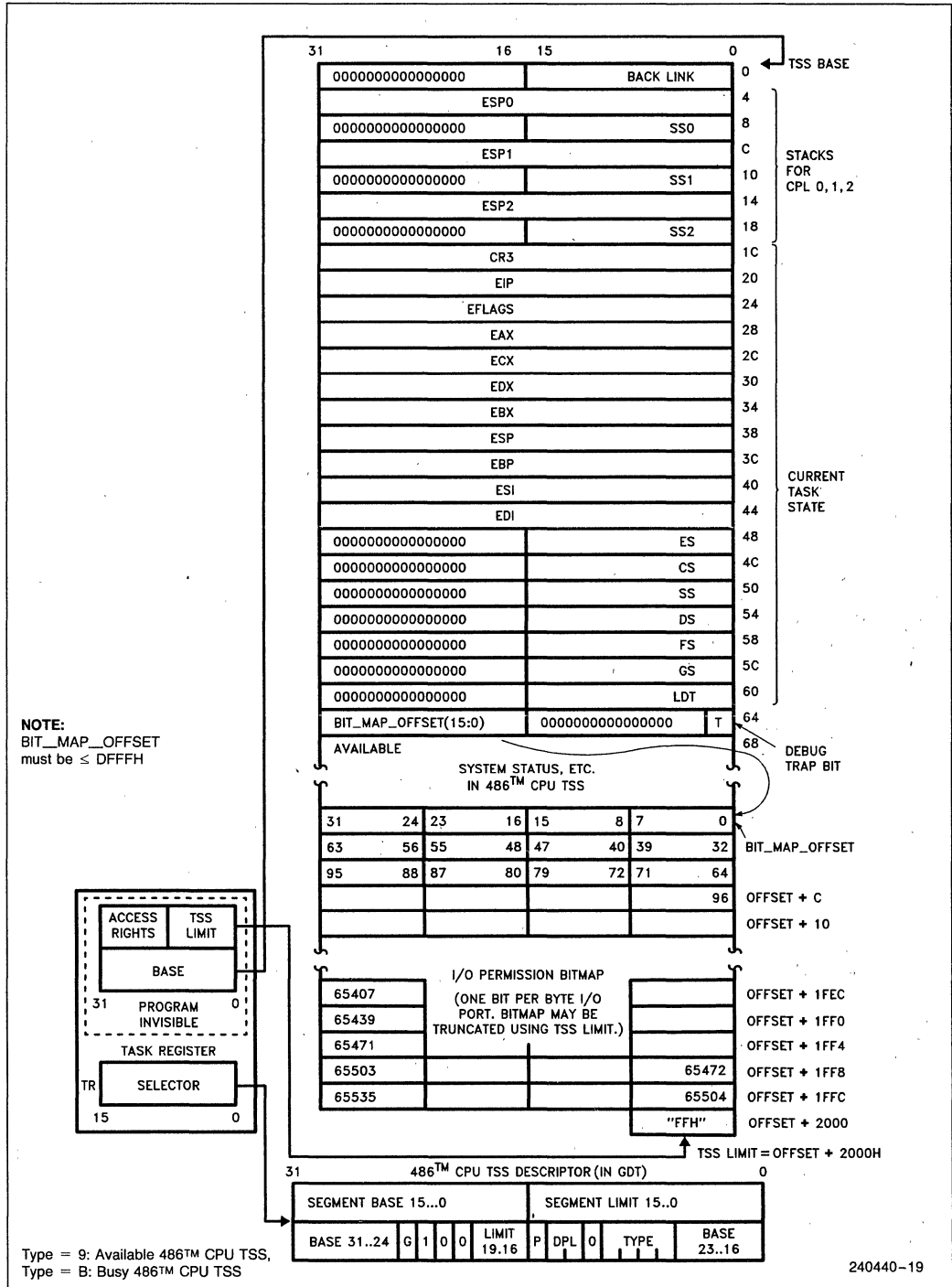


Figure 4.15a. i486™ Microprocessor TSS and TSS Registers

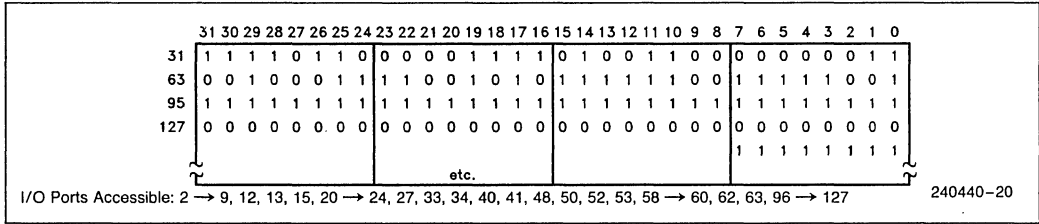


Figure 4.15b. Sample I/O Permission Bit Map

4.4.5 CALL GATES

Gates provide protected, indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures (such as those which allocate memory, or perform I/O).

Gate descriptors follow the data access rules of privilege; that is, gates can be accessed by a task if the EPL, is equal to or more privileged than the gate descriptor's DPL. Gates follow the control transfer rules of privilege and therefore may only transfer control to a more privileged level.

Call Gates are accessed via a CALL instruction and are syntactically identical to calling a normal subroutine. When an inter-level 486 Microprocessor call gate is activated, the following actions occur.

1. Load CS:EIP from gate check for validity
2. SS is pushed zero-extended to 32 bits
3. ESP is pushed
4. Copy Word Count 32-bit parameters from the old stack to the new stack
5. Push Return address on stack

The procedure is identical for 80286 Call gates, except that 16-bit parameters are copied and 16-bit registers are pushed.

Interrupt Gates and Trap gates work in a similar fashion as the call gates, except there is no copying of parameters. The only difference between Trap and Interrupt gates is that control transfers through an Interrupt gate disable further interrupts (i.e. the IF bit is set to 0), and Trap gates leave the interrupt status unchanged.

4.4.6 TASK SWITCHING

A very important attribute of any multi-tasking/multi-user operating systems is its ability to rapidly switch between tasks or processes. The 486 Microprocessor directly supports this operation by providing a task switch instruction in hardware. The 486 Microprocessor task switch operation saves the entire

state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task, in about 17 microseconds. Like transfer of control via gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 4.15) containing the entire 486 Microprocessor execution state while a task gate descriptor contains a TSS selector. The 486 Microprocessor supports both 80286 and 486 Microprocessor style TSSs. Figure 4.16 shows a 80286 TSS. The limit of a 486 Microprocessor TSS must be greater than 0064H (002BH for a 80286 TSS), and can be as large as 4 Gigabytes. In the additional TSS space, the operating system is free to store additional information such as the reason the task is inactive, time the task has spent running, and open files belong to the task.

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 486 Microprocessor called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The current executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CR0) give information about the state of a task which are useful to the operating system. The Nested Task (NT) (bit 14 in EFLAGS) controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular return; when NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

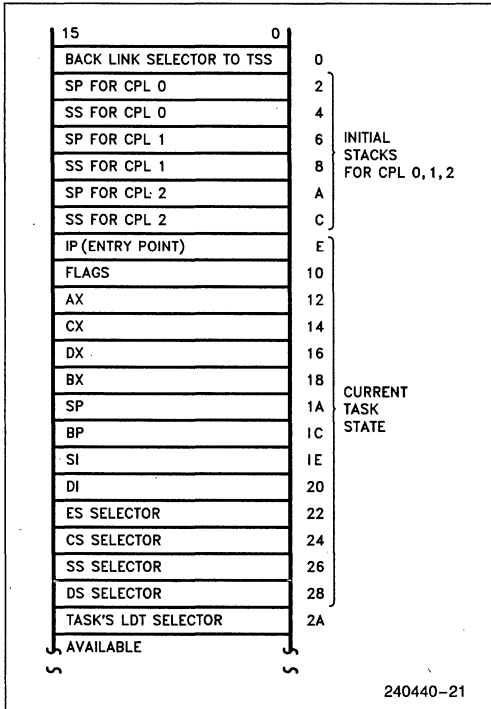


Figure 4.16. 80286 TSS

When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. (The NT bit will be restored after execution of the interrupt handler) NT may also be set or cleared by POPF or IRET instructions.

The 486 Microprocessor task state segment is marked busy by changing the descriptor type field from TYPE 9H to TYPE BH. An 80286 TSS is marked busy by changing the descriptor type field from TYPE 1 to TYPE 3. Use of a selector that references a busy task state segment causes an exception 13.

The Virtual Mode (VM) bit 17 is used to indicate if a task, is a virtual 8086 task. If VM = 1, then the tasks will use the Real Mode addressing mechanism. The virtual 8086 environment is only entered and exited via a task switch (see Section 4.6 **Virtual Mode**).

The FPU's state is not automatically saved when a task switch occurs, because the incoming task may not use the FPU. The Task Switched (TS) Bit (bit 3 in the CR0) helps deal with the FPU's state in a multi-tasking environment. Whenever the 486 Micro-

processor switches tasks, it sets the TS bit. The 486 Microprocessor detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the FPU. A processor extension not present exception (7) will occur when attempting to execute a Floating Point or WAIT instruction if the Task Switched and Monitor coprocessor extension bits are both set (i.e. TS = 1 and MP = 1).

The T bit in the 486 Microprocessor TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1 then upon entry to a new task a debug exception 1 will be generated.

4.4.7 INITIALIZATION AND TRANSITION TO PROTECTED MODE

Since the 486 Microprocessor begins executing in Real Mode immediately after RESET it is necessary to initialize the system tables and registers with the appropriate values.

The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256 bytes long, and GDT must contain descriptors for the initial code, and data segments. Figure 4.17 shows the tables and Figure 4.18 the descriptors needed for a simple Protected Mode 486 Microprocessor system. It has a single code and single data/stack segment each four gigabytes long and a single privilege level PL = 0.

The actual method of enabling Protected Mode is to load CR0 with the PE bit set, via the MOV CR0, R/M instruction. This puts the 486 Microprocessor in Protected Mode.

After enabling Protected Mode, the next instruction should execute an intersegment JMP to load the CS register and flush the instruction decode queue. The final step is to load all of the data segment registers with the initial selector values.

An alternate approach to entering Protected Mode which is especially appropriate for multi-tasking operating systems, is to use the built in task-switch to load all of the registers. In this case the GDT would contain two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode would jump to the TSS causing a task switch and loading all of the registers with the values stored in the TSS. The Task State Segment Register should be initialized to point to a valid TSS descriptor since a task switch saves the state of the current task in a task state segment.

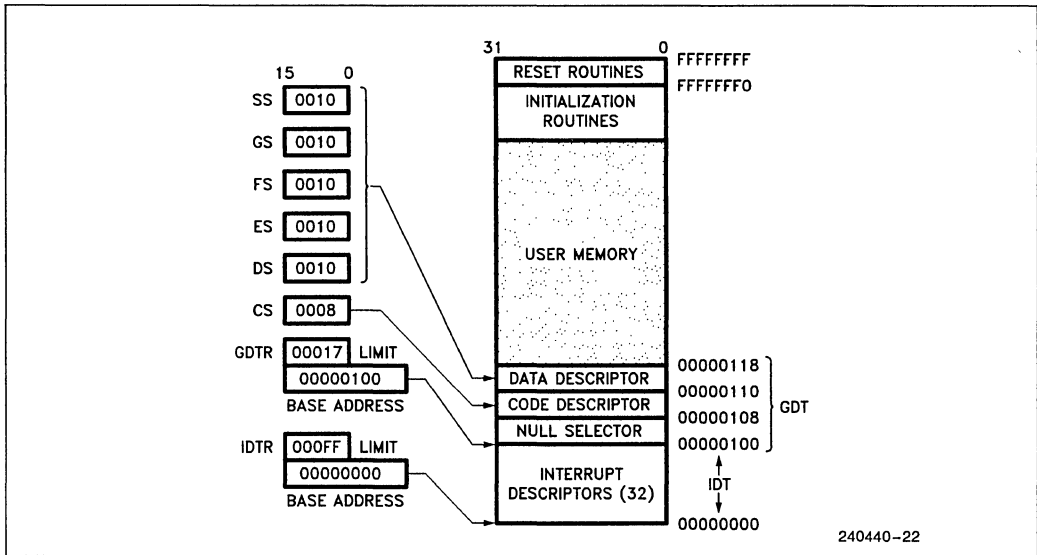


Figure 4.17. Simple Protected System

DATA DESCRIPTOR	2	BASE 31...24 00 (H)	G 1	D 1	0	0	LIMIT 19.16 F (H)	1	0	0	1	0	0	1	0	BASE 23...16 00 (H)
	SEGMENT BASE 15...0 0118 (H)							SEGMENT LIMIT 15...0 FFFF (H)								
CODE DESCRIPTOR	1	BASE 31...24 00 (H)	G 1	D 1	0	0	LIMIT 19.16 F (H)	1	0	0	1	1	0	1	0	BASE 23...16 00 (H)
	SEGMENT BASE 15...0 0118 (H)							SEGMENT LIMIT 15...0 FFFF (H)								
	NULL							DESCRIPTOR								
	0															
		31	24			16	15	8			0					

Figure 4.18. GDT Descriptors for Simple System

4.4.8 TOOLS FOR BUILDING PROTECTED SYSTEMS

In order to simplify the design of a protected multi-tasking system, Intel provides a tool which allows the system designer an easy method of constructing the data structures needed for a Protected Mode 486 Microprocessor system. This tool is the builder BLD-386™. BLD-386 lets the operating system writer specify all of the segment descriptors discussed in the previous sections (LDTs, IDTs, GDTs, Gates, and TSSs) in a high-level language.

4.5 Paging

4.5.1 PAGING CONCEPTS

Paging is another type of memory management useful for virtual memory multitasking operating systems. Unlike segmentation which modularizes programs and data into variable length segments, paging divides programs into multiple uniform size pages. Pages bear no direct relation to the logical

structure of a program. While segment selectors can be considered the logical “name” of a program module or data structure, a page most likely corresponds to only a portion of a module or data structure.

By taking advantage of the locality of reference displayed by most programs, only a small number of pages from each active task need be in memory at any one moment.

4.5.2 PAGING ORGANIZATION

4.5.2.1 Page Mechanism

The 486 Microprocessor uses two levels of tables to translate the linear address (from the segmentation unit) into a physical address. There are three components to the paging mechanism of the 486 Microprocessor: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the 486 Microprocessor paging mechanism are the same size, namely, 4 Kbytes. A uniform size for all of the elements simplifies memory allocation and reallocation schemes, since there is no problem with memory fragmentation. Figure 4.19 shows how the paging mechanism works.

4.5.2.2 Page Descriptor Base Register

CR2 is the Page Fault Linear Address register. It holds the 32-bit linear address which caused the last page fault detected.

CR3 is the Page Directory Physical Base Address Register. It contains the physical starting address of the Page Directory. The lower 12 bits of CR3 are always zero to ensure that the Page Directory is always page aligned. Loading it via a MOV CR3, reg instruction causes the Page Table Entry cache to be flushed, as will a task switch through a TSS which **changes** the value of CR0. (See 4.5.5 Translation Lookaside Buffer).

4.5.2.3 Page Directory

The Page Directory is 4 Kbytes long and allows up to 1024 Page Directory Entries. Each Page Directory Entry contains the address of the next level of tables, the Page Tables and information about the page table. The contents of a Page Directory Entry are shown in Figure 4.20. The upper 10 bits of the linear address (A22–A31) are used as an index to select the correct Page Directory Entry.

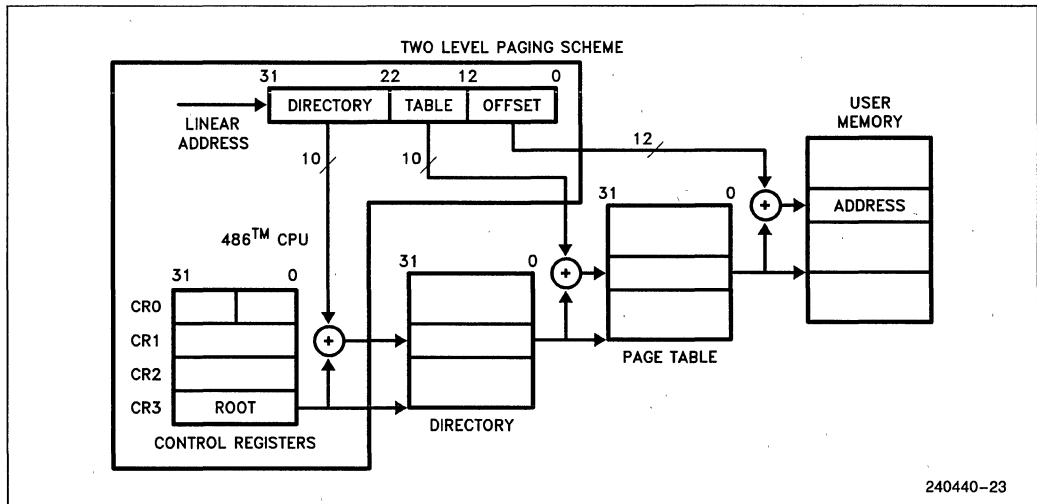


Figure 4.19. Paging Mechanism

31		12	11	10	9	8	7	6	5	4	3	2	1	0
PAGE TABLE ADDRESS 31..12			OS RESERVED			0	0	D	A	P C D	P W T	U — S	R — W	P

Figure 4.20. Page Directory Entry (Points to Page Table)

	31		12	11	10	9	8	7	6	5	4	3	2	1	0
PAGE FRAME ADDRESS 31..12	OS RESERVED			0	0	D	A	P C D	P W T	U — S	R — W	P			

Figure 4.21. Page Table Entry (Points to Page)

4.5.2.4 Page Tables

Each Page Table is 4 Kbytes and holds up to 1024 Page Table Entries. Page Table Entries contain the starting address of the page frame and statistical information about the page (see Figure 4.21). Address bits A12–A21 are used as an index to select one of the 1024 Page Table Entries. The 20 upper-bit page frame address is concatenated with the lower 12 bits of the linear address to form the physical address. Page tables can be shared between tasks and swapped to disks.

4.5.2.5 Page Directory/Table Entries

The lower 12 bits of the Page Table Entries and Page Directory Entries contain statistical information about pages and page tables respectively. The **P** (Present) bit 0 indicates if a Page Directory or Page Table entry can be used in address translation. If $P = 1$ the entry can be used for address translation if $P = 0$ the entry can not be used for translation, and all of the other bits are available for use by the software. For example the remaining 31 bits could be used to indicate where on the disk the page is stored.

The **A** (Accessed) bit 5, is set by the 486 Microprocessor for both types of entries before a read or write access occurs to an address covered by the entry. The **D** (Dirty) bit 6 is set to 1 before a write to an address covered by that page table entry occurs. The D bit is undefined for Page Directory Entries. When the P, A and D bits are updated by the 486 Microprocessor, the processor generates a Read-Modify-Write cycle which locks the bus and prevents conflicts with other processors or peripherals. Software which modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multi-master systems.

The 3 bits marked **OS Reserved** in Figure 4.20 and Figure 4.21 (bits 9–11) are software definable. OSs are free to use these bits for whatever purpose they wish. An example use of the **OS Reserved** bits would be to store information about page aging. By keeping track of how long a page has been in memory since being accessed, an operating system can implement a page replacement algorithm like Least Recently Used.

The (User/Supervisor) U/S bit 2 and the (Read/Write) R/W bit 1 are used to provide protection attributes for individual pages.

4.5.3 PAGE LEVEL PROTECTION (R/W, U/S BITS)

The 486 microprocessor provides a set of protection attributes for paging systems. The paging mechanism distinguishes between two levels of protection: User which corresponds to level 3 of the segmentation based protection, and supervisor which encompasses all of the other protection levels (0, 1, 2).

The R/W and U/S bits are used in conjunction with the WP bit in the flags register (EFLAGS). The 386 microprocessor does not contain the WP bit. The WP bit has been added to the 486 microprocessor to protect read-only pages from supervisor write accesses. The 386 microprocessor allows a read-only page to be written from protection levels 0, 1 or 2. $WP = 0$ is the 386 microprocessor compatible mode. When $WP = 0$ the supervisor can write to a read-only page as defined by the U/S and R/W bits. When $WP = 1$ supervisor access to a read-only page ($R/W = 0$) will cause a page fault (exception 14).

Table 4.4 shows the affect of the WP, U/S and R/W bits on accessing memory. When $WP = 0$, the supervisor can write to pages regardless of the state of the R/W bit. When $WP = 1$ and $R/W = 0$ the supervisor cannot write to a read-only page. A user attempt to access a supervisor only page ($U/S = 0$), or write to a read only page will cause a page fault (exception 14).

The R/W and U/S bits provide protection from user access on a page by page basis since the bits are contained in the Page Table Entry and the Page Directory Table. The U/S and R/W bits in the first level Page Directory Table apply to all entries in the page table pointed to by that directory entry. The U/S and R/W bits in the second level Page Table Entry apply only to the page described by that entry. The most restrictive of the U/S and R/W bits from the Page Directory Table and the Page Table Entry are used to address a page.

Example: If the U/S and R/W bits for the Page Directory entry were 10 (user read/execute) and the

U/S and R/W bits for the Page Table Entry were 01 (no user access at all), the access rights for the page would be 01, the numerically smaller of the two.

Note that a given segment can be easily made read-only for level 0, 1 or 2 via use of segmented protection mechanisms. (Section 4.4 **Protection**).

4.5.4 PAGE CACHEABILITY (PWT AND PCD BITS)

PWT (page write through) and PCD (page cache disable) are two new bits defined in entries in both levels of the page table structure, the Page Directory Table and the Page Table Entry. PCD and PWT control page cacheability and write policy.

PWT controls write policy. PWT = 1 defines a write-through policy for the current page. PWT = 0 allows the possibility of write-back. PWT is ignored internally because the 486 microprocessor has a write-through cache. PWT can be used to control the write policy of a second level cache.

PCD controls cacheability. PCD = 0 enables caching in the on-chip cache. PCD alone does not enable caching, it must be conditioned by the KEN# (cache enable) input signal and the state of the CD (cache disable bit) and NW (no write-through) bits in control register 0 (CR0). When PCD = 1, caching is disabled regardless of the state of KEN#, CD and NW. (See Section 5.0, **On-Chip Cache**).

The state of the PCD and PWT bits are driven out on the PCD and PWT pins during a memory access.

The PWT and PCD bits for a bus cycle are obtained either from control register 3 (CR3), the Page Directory Entry or the Page Table Entry, depending on the type of cycle run. If paging is not enabled (PG = 0 in CR0), or for cycles which bypass paging (i.e., I/O (input/output) references, INTR (interrupt request) and Halt cycles), the PWT and PCD bits are taken

from bits 3 and 4 of CR3. These bits in CR3 are initialized to zero at reset, but can be set to any value by level 0 software.

When paging is enabled (PG = 1 in CR0), the bits from the page table entry are cached in the translation lookaside buffer (TLB), and are driven any time the page mapped by the TLB entry is referenced. For normal memory cycles run with paging enabled, the PWT and PCD bits are taken from the Page Table Entry. During TLB refresh cycles when the Page Directory and Page Table entries are read, the PWT and PCD bits must be obtained elsewhere. The bits are taken from CR3 when a Page Directory Entry is being read. The bits are taken from the Page Directory Entry when the Page Table Entry is being updated.

4.5.5 TRANSLATION LOOKASIDE BUFFER

The 486 Microprocessor paging hardware is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the processor was required to access two levels of tables for every memory reference. To solve this problem, the 486 Microprocessor keeps a cache of the most recently accessed pages, this cache is called the Translation Lookaside Buffer (TLB). The TLB is a four-way set associative 32-entry page table cache. It automatically keeps the most commonly used Page Table Entries in the processor. The 32-entry TLB coupled with a 4K page size, results in coverage of 128 Kbytes of memory addresses. For many common multi-tasking systems, the TLB will have a hit rate of about 98%. This means that the processor will only have to access the two-level page structure on 2% of all memory references. Figure 4.22 illustrates how the TLB complements the 486 Microprocessor's paging mechanism.

Reading a new entry into the TLB (TLB refresh) is a two step process handled by the 486 microprocessor hardware. The sequence of data cycles to perform a TLB refresh are:

Table 4.4. Page Level Protection Attributes

U/S	R/W	WP	User Access	Supervisor Access
0	0	0	None	Read/Write/Execute
0	1	0	None	Read/Write/Execute
1	0	0	Read/Execute	Read/Write/Execute
1	1	0	Read/Write/Execute	Read/Write/Execute
0	0	1	None	Read/Execute
0	1	1	None	Read/Write/Execute
1	0	1	Read/Execute	Read/Execute
1	1	1	Read/Write/Execute	Read/Write/Execute

1. Read the correct Page Directory Entry, as pointed to by the page base register and the upper 10 bits of the linear address. The page base register is in control register 3.
 - 1a. Optionally perform a locked read/write to set the accessed bit in the directory entry. The directory entry will actually get read twice if the 486 microprocessor needs to set any of the bits in the entry. If the page directory entry changes between the first and second reads, the data returned for the second read will be used.
 2. Read the correct entry in the Page Table and place the entry in the TLB.
 - 2a. Optionally perform a locked read/write to set the accessed and/or dirty bit in the page table entry. Again, note that the page table entry will actually get read twice if the 486 microprocessor needs to set any of the bits in the entry. Like the directory entry, if the data changes between the first and second read the data returned for the second read will be used.

Note that the directory entry must always be read into the processor, since directory entries are never placed in the paging TLB. Page faults can be signaled from either the page directory read or the page table read. Page directory and page table entries may be placed in the 486 on-chip cache just like normal data.

4.5.6 PAGING OPERATION

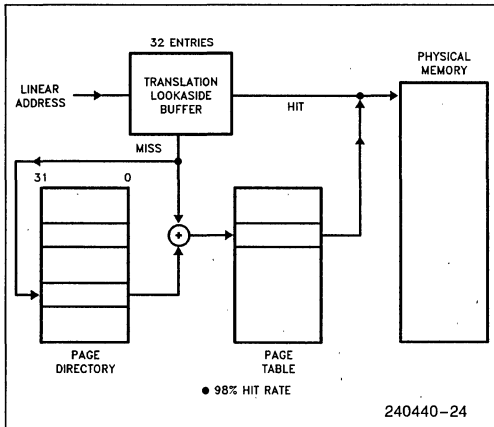


Figure 4.22. Translation Lookaside Buffer

The paging hardware operates in the following fashion. The paging unit hardware receives a 32-bit linear address from the segmentation unit. The upper 20 linear address bits are compared with all 32 entries in the TLB to determine if there is a match. If there is a match (i.e., a TLB hit), then the 32-bit physical address is calculated and will be placed on the address bus.

However, if the page table entry is not in the TLB, the 486 Microprocessor will read the appropriate Page Directory Entry. If $P = 1$ on the Page Directory Entry indicating that the page table is in memory, then the 486 Microprocessor will read the appropriate Page Table Entry and set the Access bit. If $P = 1$ on the Page Table Entry indicating that the page is in memory, the 486 Microprocessor will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. However, if $P = 0$ for either the Page Directory Entry or the Page Table Entry, then the processor will generate a page fault, an Exception 14.

The processor will also generate an exception 14 page fault, if the memory reference violated the page protection attributes (i.e., U/S or R/W) (e.g., trying to write to a read-only page). CR2 will hold the linear address which caused the page fault. If a second page fault occurs, while the processor is attempting to enter the service routine for the first, then the processor will invoke the page fault (exception 14) handler a second time, rather than the double fault (exception 8) handler. Since Exception 14 is classified as a fault, CS: EIP will point to the instruction causing the page fault. The 16-bit error code pushed as part of the page fault handler will contain status bits which indicate the cause of the page fault.

The 16-bit error code is used by the operating system to determine how to handle the page fault. Figure 4.23a shows the format of the page-fault error code and the interpretation of the bits.

NOTE:

Even though the bits in the error code (U/S, W/R, and P) have similar names as the bits in the Page Directory/Table Entries, the interpretation of the error code bits is different. Figure 4.23b indicates what type of access caused the page fault.

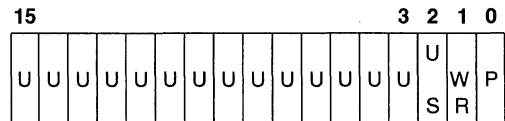


Figure 4.23a. Page Fault Error Code Format

U/S: The U/S bit indicates whether the access causing the fault occurred when the processor was executing in User Mode ($U/S = 1$) or in Supervisor mode ($U/S = 0$).

W/R: The W/R bit indicates whether the access causing the fault was a Read ($W/R = 0$) or a Write ($W/R = 1$).

P: The P bit indicates whether a page fault was caused by a not-present page ($P = 0$), or by a page level protection violation ($P = 1$).

U: UNDEFINED

U/S	W/R	Access Type
0	0	Supervisor* Read
0	1	Supervisor Write
1	0	User Read
1	1	User Write

*Descriptor table access will fault with U/S = 0, even if the program is executing at level 3.

Figure 4.23b. Type of Access Causing Page Fault

4.5.7 OPERATING SYSTEM RESPONSIBILITIES

The 486 Microprocessor takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for setting up the initial page tables, and handling any page faults. The operating system also is required to invalidate (i.e., flush) the TLB when any changes are made to any of the page table entries. The operating system must reload CR3 to cause the TLB to be flushed.

Setting up the tables is simply a matter of loading CR3 with the address of the Page Directory, and allocating space for the Page Directory and the Page Tables. The primary responsibility of the operating system is to implement a swapping policy and handle all of the page faults.

A final concern of the operating system is to ensure that the TLB cache matches the information in the paging tables. In particular, any time the operating system sets the P present bit of page table entry to zero, the TLB must be flushed. Operating systems may want to take advantage of the fact that CR3 is stored as part of a TSS, to give every task or group of tasks its own set of page tables.

4.6 Virtual 8086 Environment

4.6.1 EXECUTING 8086 PROGRAMS

The 486 Microprocessor allows the execution of 8086 application programs in both Real Mode and in the Virtual 8086 Mode (Virtual Mode). Of the two methods, Virtual 8086 Mode offers the system designer the most flexibility. The Virtual 8086 Mode allows the execution of 8086 applications, while still allowing the system designer to take full advantage of the 486 Microprocessor protection mechanism. In

particular, the 486 Microprocessor allows the simultaneous execution of 8086 operating systems and its applications, and a 486 Microprocessor operating system and both 80286 and 486 Microprocessor applications. Thus, in a multi-user 486 Microprocessor computer, one person could be running an MS-DOS spreadsheet, another person using MS-DOS, and a third person could be running multiple Unix utilities and applications. Each person in this scenario would believe that he had the computer completely to himself. Figure 4.24 illustrates this concept.

4.6.2 VIRTUAL 8086 MODE ADDRESSING MECHANISM

One of the major differences between 486 Microprocessor Real and Protected modes is how the segment selectors are interpreted. When the processor is executing in Virtual 8086 Mode the segment registers are used in an identical fashion to Real Mode. The contents of the segment register is shifted left 4 bits and added to the offset to form the segment base linear address.

The 486 Microprocessor allows the operating system to specify which programs use the 8086 style address mechanism, and which programs use Protected Mode addressing, on a per task basis. Through the use of paging, the one megabyte address space of the Virtual Mode task can be mapped to anywhere in the 4 gigabyte linear address space of the 486 Microprocessor. Like Real Mode, Virtual Mode effective addresses (i.e., segment offsets) that exceed 64 Kbyte will cause an exception 13. However, these restrictions should not prove to be important, because most tasks running in Virtual 8086 Mode will simply be existing 8086 application programs.

4.6.3 PAGING IN VIRTUAL MODE

The paging hardware allows the concurrent running of multiple Virtual Mode tasks, and provides protection and operating system isolation. Although it is not strictly necessary to have the paging hardware enabled to run Virtual Mode tasks, it is needed in order to run multiple Virtual Mode tasks or to relocate the address space of a Virtual Mode task to physical address space greater than one megabyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into up to 256 pages. Each one of the pages can be located anywhere within the maximum 4 gigabyte physical address space of the 486 Microprocessor. In addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations.

Finally, the paging hardware allows the sharing of the 8086 operating system code between multiple 8086 applications. Figure 4.24 shows how the 486 Microprocessor paging hardware enables multiple 8086 programs to run under a virtual memory demand paged system.

4.6.4 PROTECTION AND I/O PERMISSION BITMAP

All Virtual 8086 Mode programs execute at privilege level 3, the level of least privilege. As such, Virtual 8086 Mode programs are subject to all of the protection checks defined in Protected Mode. (This is different from Real Mode which implicitly is executing at privilege level 0, the level of greatest privilege.) Thus, an attempt to execute a privileged instruction when in Virtual 8086 Mode will cause an exception 13 fault.

The following are privileged instructions, which may be executed only at Privilege Level 0. Therefore, attempting to execute these instructions in Virtual 8086 Mode (or anytime CPL > 0) causes an exception 13 fault:

```
LIDT; MOV DRn,reg; MOV reg,DRn;
LGDT; MOV TRn,reg; MOV reg,TRn;
LMSW; MOV CRn,reg; MOV reg,CRn.
CLTS;
HLT;
```

Several instructions, particularly those applying to the multitasking model and protection model, are available only in Protected Mode. Therefore, attempting to execute the following instructions in Real Mode or in Virtual 8086 Mode generates an exception 6 fault:

```
LTR; STR;
LLDT; SLDT;
LAR; VERR;
LSL; VERW;
ARPL.
```

The instructions which are IOPL-sensitive in Protected Mode are:

```
IN; STI;
OUT; CLI
INS;
OUTS;
REP INS;
REP OUTS;
```

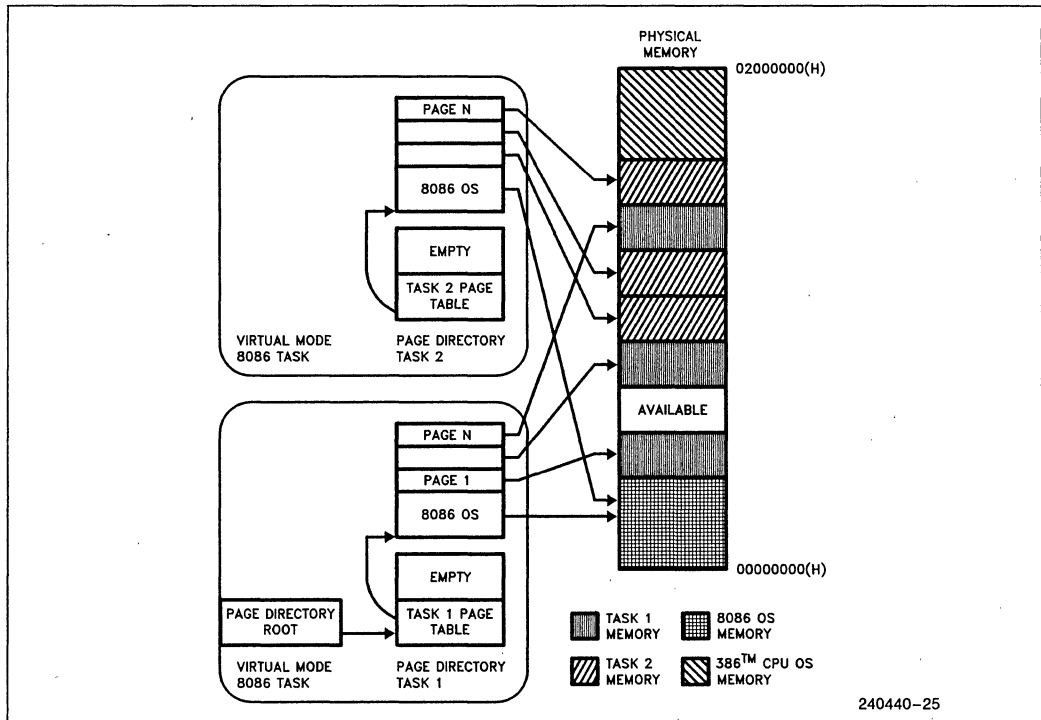


Figure 4.24. Virtual 8086 Environment Memory Management

In Virtual 8086 Mode, a slightly different set of instructions are made IOPL-sensitive. The following instructions are IOPL-sensitive in Virtual 8086 Mode:

```
INT n;   STI;
PUSHF;  CLI;
POPF;   IRET
```

The PUSHF, POPF, and IRET instructions are IOPL-sensitive in Virtual 8086 Mode only. This provision allows the IF flag (interrupt enable flag) to be virtualized to the Virtual 8086 Mode program. The INT n software interrupt instruction is also IOPL-sensitive in Virtual 8086 Mode. Note, however, that the INT 3 (opcode 0CCH), INTO, and BOUND instructions are not IOPL-sensitive in Virtual 8086 mode (they aren't IOPL sensitive in Protected Mode either).

Note that the I/O instructions (IN, OUT, INS, OUTS, REP INS, and REP OUTS) are **not** IOPL-sensitive in Virtual 8086 mode. Rather, the I/O instructions become automatically sensitive to the **I/O Permission Bitmap** contained in the **486 Microprocessor Task State Segment**. The I/O Permission Bitmap, automatically used by the 486 Microprocessor in Virtual 8086 Mode, is illustrated by Figures 4.15a and 4.15b.

The I/O Permission Bitmap can be viewed as a 0–64 Kbit bit string, which begins in memory at offset `Bit_Map_Offset` in the current TSS. `Bit_Map_Offset` must be \leq DFFFH so the entire bit map and the byte FFH which follows the bit map are all at offsets \leq FFFFH from the TSS base. The 16-bit pointer `Bit_Map_Offset` (15:0) is found in the word beginning at offset 66H (102 decimal) from the TSS base, as shown in Figure 4.15a.

Each bit in the I/O Permission Bitmap corresponds to a single byte-wide I/O port, as illustrated in Figure 4.15a. If a bit is 0, I/O to the corresponding byte-wide port can occur without generating an exception. Otherwise the I/O instruction causes an exception 13 fault. Since every byte-wide I/O port must be protectable, all bits corresponding to a word-wide or dword-wide port must be 0 for the word-wide or dword-wide I/O to be permitted. If all the referenced bits are 0, the I/O will be allowed. If any referenced bits are 1, the attempted I/O will cause an exception 13 fault.

Due to the use of a pointer to the base of the I/O Permission Bitmap, the bitmap may be located anywhere within the TSS, or may be ignored completely by pointing the `Bit_Map_Offset` (15:0) beyond the limit of the TSS segment. In the same manner, only a small portion of the 64K I/O space need have an associated map bit, by adjusting the TSS limit to truncate the bitmap. This eliminates the commitment of 8K of memory when a complete bitmap is not required, while allowing the fully general case if desired.

EXAMPLE OF BITMAP FOR I/O PORTS 0–255: Setting the TSS limit to `{bit_Map_Offset + 31 + 1**}` [**** see note below**] will allow a 32-byte bitmap for the I/O ports #0–255, plus a terminator byte of all 1's [**** see note below**]. This allows the I/O bitmap to control I/O Permission to I/O port 0–255 while causing an exception 13 fault on attempted I/O to any I/O port 80256 through 65,565.

****IMPORTANT IMPLEMENTATION NOTE:** Beyond the last byte of I/O mapping information in the I/O Permission Bitmap **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the 486 Microprocessor TSS segment (see Figure 4.15a).

4.6.5 INTERRUPT HANDLING

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled in a unique fashion. When running in Virtual Mode all interrupts and exceptions involve a privilege change back to the host 486 Microprocessor operating system. The 486 Microprocessor operating system determines if the interrupt comes from a Protected Mode application or from a Virtual Mode program by examining the VM bit in the EFLAGS image stored on the stack.

When a Virtual Mode program is interrupted and execution passes to the interrupt routine at level 0, the VM bit is cleared. However, the VM bit is still set in the EFLAG image on the stack.

The 486 Microprocessor operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The 486 Microprocessor operating system may choose to let the 8086 operating system handle the interrupt or it may emulate the function of the interrupt handler. For example, many 8086 operating system calls are accessed by PUSHing parameters on the stack, and then executing an INT n instruction. If the IOPL is set to 0 then all INT n instructions will be intercepted by the 486 Microprocessor operating system. The 486 Microprocessor operating system could emulate the 8086 operating system's call. Figure 4.25 shows how the 486 Microprocessor operating system could intercept an 8086 operating system's call to "Open a File".

A 486 Microprocessor operating system can provide a Virtual 8086 Environment which is totally transparent to the application software via intercepting and then emulating 8086 operating system's calls, and intercepting IN and OUT instructions.

4.6.6 ENTERING AND LEAVING VIRTUAL 8086 MODE

Virtual 8086 mode is entered by executing an IRET instruction (at CPL=0), or Task Switch (at any CPL) to a 486 Microprocessor task whose 486 Microprocessor TSS has a FLAGS image containing a 1 in the VM bit position while the processor is executing in Protected Mode. That is, one way to enter Virtual 8086 mode is to switch to a task with a 486 Microprocessor TSS that has a 1 in the VM bit in the EFLAGS image. The other way is to execute a 32-bit IRET instruction at privilege level 0, where the stack has a 1 in the VM bit in the EFLAGS image. POPF does not affect the VM bit, even if the processor is in Protected Mode or level 0, and so cannot be used to enter Virtual 8086 Mode. PUSHF always pushes a 0 in the VM bit, even if the processor is in Virtual 8086 Mode, so that a program cannot tell if it is executing in REAL mode, or in Virtual 8086 mode.

The VM bit can be set by executing an IRET instruction only at privilege level 0, or by any instruction or Interrupt which causes a task switch in Protected Mode (with VM=1 in the new FLAGS image), and can be cleared only by an interrupt or exception in Virtual 8086 Mode. IRET and POPF instructions executed in REAL mode or Virtual 8086 mode will not change the value in the VM bit.

The transition out of virtual 8086 mode to 486 Microprocessor protected mode occurs only on receipt of an interrupt or exception (such as due to a sensitive instruction). In Virtual 8086 mode, all interrupts and exceptions vector through the protected mode IDT, and enter an interrupt handler in protected 486 Microprocessor mode. That is, as part of interrupt processing, the VM bit is cleared.

Because the matching IRET must occur from level 0, if an Interrupt or Trap Gate is used to field an interrupt or exception out of Virtual 8086 mode, the Gate must perform an inter-level interrupt only to level 0. Interrupt or Trap Gates through conforming segments, or through segments with DPL>0, will raise a GP fault with the CS selector as the error code.

4.6.6.1 Task Switches To/From Virtual 8086 Mode

Tasks which can execute in virtual 8086 mode must be described by a TSS with the new 486 Microprocessor format (TYPE 9 or 11 descriptor).

A task switch out of virtual 8086 mode will operate exactly the same as any other task switch out of a task with a 486 Microprocessor TSS. All of the programmer visible state, including the FLAGS register with the VM bit set to 1, is stored in the TSS.

The segment registers in the TSS will contain 8086 segment base values rather than selectors.

A task switch into a task described by a 486 Microprocessor TSS will have an additional check to determine if the incoming task should be resumed in virtual 8086 mode. Tasks described by 80286 format TSSs cannot be resumed in virtual 8086 mode, so no check is required there (the FLAGS image in 80286 format TSS has only the low order 16 FLAGS bits). Before loading the segment register images from a 486 Microprocessor TSS, the FLAGS image is loaded, so that the segment registers are loaded from the TSS image as 8086 segment base values. The task is now ready to resume in virtual 8086 execution mode.

4.6.6.2 Transitions Through Trap and Interrupt Gates, and IRET

A task switch is one way to enter or exit virtual 8086 mode. The other method is to exit through a Trap or Interrupt gate, as part of handling an interrupt, and to enter as part of executing an IRET instruction. The transition out must use a 486 Microprocessor Trap Gate (Type 14), or 486 Microprocessor Interrupt Gate (Type 15), which must point to a non-conforming level 0 segment (DPL=0) in order to permit the trap handler to IRET back to the Virtual 8086 program. The Gate must point to a non-conforming level 0 segment to perform a level switch to level 0 so that the matching IRET can change the VM bit. 486 Microprocessor gates must be used, since 80286 gates save only the low 16 bits of the FLAGS register, so that the VM bit will not be saved on transitions through the 80286 gates. Also, the 16-bit IRET (presumably) used to terminate the 80286 interrupt handler will pop only the lower 16 bits from FLAGS, and will not affect the VM bit. The action taken for a 486 Microprocessor Trap or Interrupt gate if an interrupt occurs while the task is executing in virtual 8086 mode is given by the following sequence.

- (1) Save the FLAGS register in a temp to push later. Turn off the VM and TF bits, and if the interrupt is serviced by an Interrupt Gate, turn off IF also.
- (2) Interrupt and Trap gates must perform a level switch from 3 (where the VM86 program executes) to level 0 (so IRET can return). This process involves a stack switch to the stack given in the TSS for privilege level 0. Save the Virtual 8086 Mode SS and ESP registers to push in a later step. The segment register load of SS will be done as a Protected Mode segment load, since the VM bit was turned off above.

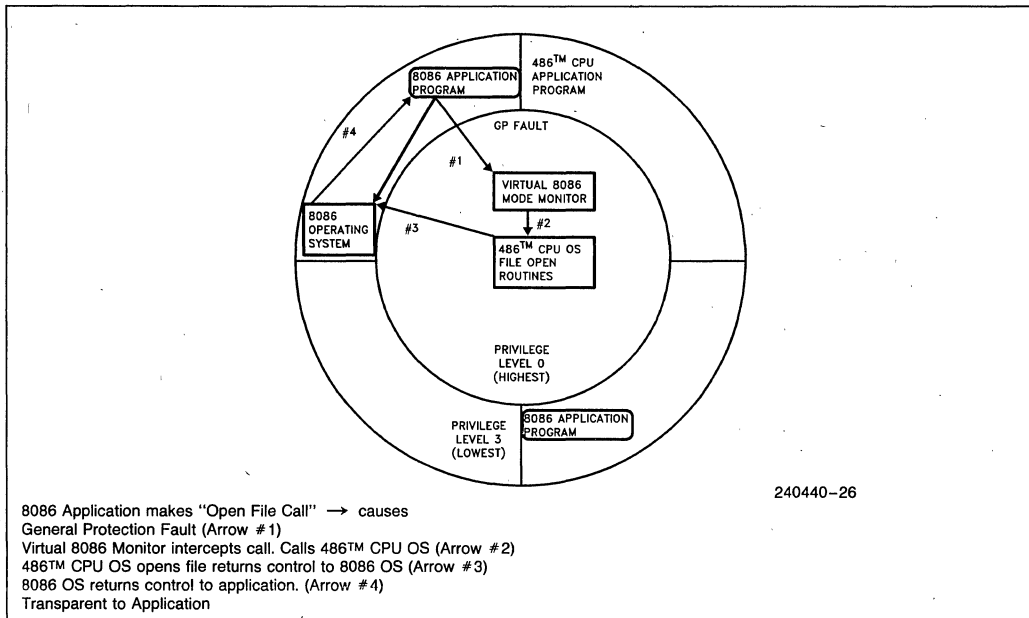


Figure 4.25. Virtual 8086 Environment Interrupt and Call Handling

- (3) Push the 8086 segment register values onto the new stack, in the order: GS, FS, DS, ES. These are pushed as 32-bit quantities, with undefined values in the upper 16 bits. Then load these 4 registers with null selectors (0).
- (4) Push the old 8086 stack pointer onto the new stack by pushing the SS register (as 32-bits, high bits undefined), then pushing the 32-bit ESP register saved above.
- (5) Push the 32-bit FLAGS register saved in step 1.
- (6) Push the old 8086 instruction pointer onto the new stack by pushing the CS register (as 32-bits, high bits undefined), then pushing the 32-bit EIP register.
- (7) Load up the new CS:EIP value from the interrupt gate, and begin execution of the interrupt routine in protected 486 Microprocessor mode.

The transition out of virtual 8086 mode performs a level change and stack switch, in addition to changing back to protected mode. In addition, all of the 8086 segment register images are stored on the stack (behind the SS:ESP image), and then loaded with null (0) selectors before entering the interrupt handler. This will permit the handler to safely save and restore the DS, ES, FS, and GS registers as 80286 selectors. This is needed so that interrupt handlers which don't care about the mode of the interrupted program can use the same prolog and epilog code for state saving (i.e., push all registers in prolog, pop all in epilog) regardless of whether or not

a "native" mode or Virtual 8086 mode program was interrupted. Restoring null selectors to these registers before executing the IRET will not cause a trap in the interrupt handler. Interrupt routines which expect values in the segment registers, or return values in segment registers will have to obtain/return values from the 8086 register images pushed onto the new stack. They will need to know the mode of the interrupted program in order to know where to find/return segment registers, and also to know how to interpret segment register values.

The IRET instruction will perform the inverse of the above sequence. Only the extended 486 Microprocessors IRET instruction (operand size = 32) can be used, and must be executed at level 0 to change the VM bit to 1.

- (1) If the NT bit in the FLAGS register is on, an inter-task return is performed. The current state is stored in the current TSS, and the link field in the current TSS is used to locate the TSS for the interrupted task which is to be resumed.

Otherwise, continue with the following sequence.

- (2) Read the FLAGS image from SS:8[ESP] into the FLAGS register. This will set VM to the value active in the interrupted routine.
- (3) Pop off the instruction pointer CS:EIP. EIP is popped first, then a 32-bit word is popped which contains the CS value in the lower 16 bits. If

VM=0, this CS load is done as a protected mode segment load. If VM=1, this will be done as an 8086 segment load.

- (4) Increment the ESP register by 4 to bypass the FLAGS image which was "popped" in step 1.
- (5) If VM=1, load segment registers ES, DS, FS, and GS from memory locations SS:[ESP+8], SS:[ESP+12], SS:[ESP+16], and SS:[ESP+20], respectively, where the new value of ESP stored in step 4 is used. Since VM=1, these are done as 8086 segment register loads.
Else if VM=0, check that the selectors in ES, DS, FS, and GS are valid in the interrupted routine. Null out invalid selectors to trap if an attempt is made to access through them.
- (6) If (RPL(CS) > CPL), pop the stack pointer SS:ESP from the stack. The ESP register is popped first, followed by 32-bits containing SS in the lower 16 bits. If VM=0, SS is loaded as a protected mode segment register load. If VM=1, an 8086 segment register load is used.
- (7) Resume execution of the interrupted routine. The VM bit in the FLAGS register (restored from the interrupt routine's stack image in step 1) determines whether the processor resumes the interrupted routine in Protected mode of Virtual 8086 mode.

software transparent to maintain binary compatibility with previous generations of the x86 architecture.

The on-chip cache has been designed for maximum flexibility and performance. The cache has several operating modes offering flexibility during program execution and debugging. Memory areas can be defined as non-cacheable by software and external hardware. Protocols for cache line invalidations and replacement are implemented in hardware, easing system design.

5.1 Cache Organization

The on-chip cache is a unified code and data cache. The cache is used for both instruction and data accesses and acts on physical addresses.

The cache organization is 4-way set associative and each line is 16 bytes wide. The eight Kbytes of cache memory are logically organized as 128 sets, each containing four lines.

The cache memory is physically split into four 2-Kbyte blocks each containing 128 lines (see Figure 5.1). Associated with each 2-Kbyte block are 128 21-bit tags. There is a valid bit for each line in the cache. Each line in the cache is either valid or not valid. There are no provisions for partially valid lines.

5.0 ON-CHIP CACHE

To meet its performance goals the 486 microprocessor contains an eight Kbyte cache. The cache is

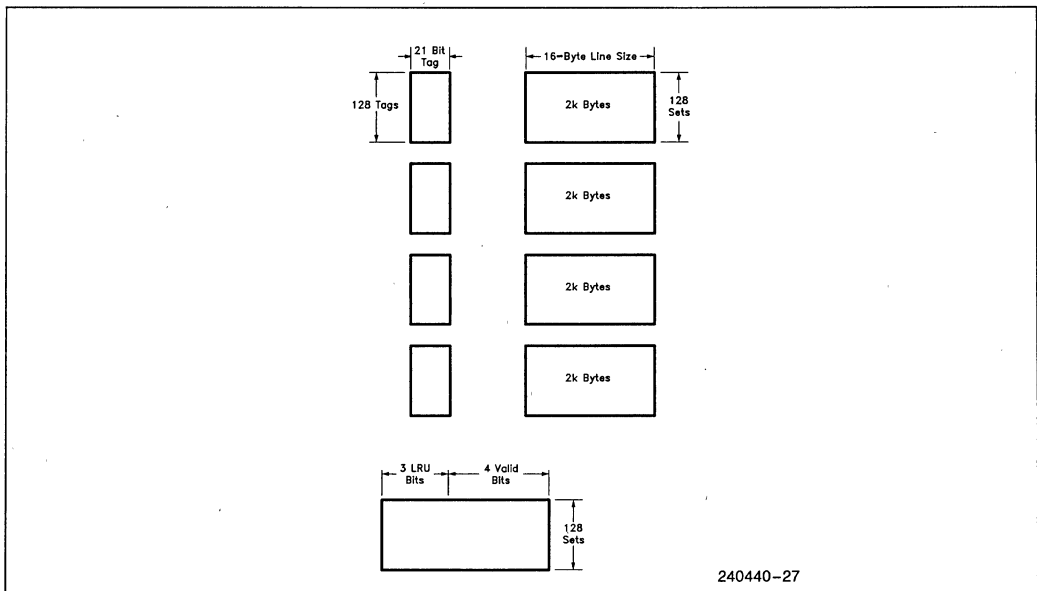


Figure 5.1. On-Chip Cache Physical Organization

The write strategy of on-chip cache is write-through. All writes will drive an external write bus cycle in addition to writing the information to the internal cache if the write was a cache hit. A write to an address not contained in the internal cache will only be written to external memory. Cache allocations are not made on write misses.

5.2 Cache Control

Control of the cache is provided by the CD and NW bits in CR0. CD enables and disables the cache. NW controls memory write-through and invalidates.

The CD and NW bits define four operating modes of the on-chip cache as given in Table 5.1. These modes provide flexibility in how the on-chip cache is used.

The CD and NW bits define four operating modes of the on-chip code and data cache, as given in the following table:

Table 5.1. Cache Operating Modes

CD	NW	Operating Mode
1	1	Cache fills disabled, write-through and invalidates disabled
1	0	Cache fills disabled, write-through and invalidates enabled
0	1	INVALID. IF CR0 is loaded with this configuration of bits, a GP fault with error code of 0 is raised.
0	0	Cache fills enabled, write-through and invalidates enabled

CD=1, NW=1

The cache is completely disabled by setting CD=1 and NW=1 and then flushing the cache. This mode may be useful for debugging programs where it is important to see all memory cycles at the pins. Writes which hit in the cache will not appear on the external bus.

It is possible to use the on-chip cache as fast static RAM by "pre-loading" certain memory areas into the cache and then setting CD=1 and NW=1. Pre-loading can be done by careful choice of memory references with the cache turned on or by use of the testability functions (see Section 8.2). When the cache is turned off the memory mapped by the cache is "frozen" into the cache since fills and invalidates are disabled.

CD=1, NW=0

Cache fills are disabled but write-throughs and invalidates are enabled. This mode is the same as if the KEN# pin was strapped HIGH disabling cache fills. Write-throughs and invalidates may still occur to keep the cache valid. This mode is useful if the software must disable the cache for a short period of time, and then re-enable it without flushing the original contents.

CD=0, NW=1

INVALID. If CR0 is loaded with this bit configuration, a General Protection fault with error code of 0 is raised. Note that this mode would imply a non-transparent write-back cache. A future processor may define this combination of bits to implement a write-back cache.

CD=0, NW=0

This is the normal operating mode.

Completely disabling the cache is a two step process. First CD and NW must be set to 1 and then the cache must be flushed. If the cache is not flushed, cache hits on reads will still occur and data will be read from the cache.

5.3 Cache Line Fills

Any area of memory can be cached in the 486 microprocessor. Non-cacheable portions of memory can be defined by the external system or by software. The external system can inform the 486 microprocessor that a memory address is non-cacheable by returning the KEN# pin inactive during a memory access (refer to Section 7.2.3). Software can prevent certain pages from being cached by setting the PCD bit in the page table entry.

A read request can be generated from program operation or by an instruction pre-fetch. The data will be supplied from the on-chip cache if a cache hit occurs on the read address. If the address is not in the cache, a read request for the data is generated on the external bus.

If the read request is to a cacheable portion of memory, the 486 microprocessor initiates a cache line fill. During a line fill a 16-byte line is read into the 486 microprocessor.

Cache fills will only be generated for read misses. Write misses will never cause a line in the internal cache to be allocated. If a cache hit occurs on a write, the line will be updated.

Cache line fills can be performed over 8- and 16-bit busses using the dynamic bus sizing feature. Refer to Section 7.1.3 for a description of dynamic bus sizing.

Refer to Section 7.2.3 for further information on cacheable cycles.

5.4 Cache Line Invalidations

The 486 microprocessor contains both a hardware and software mechanism for invalidating lines in its internal cache. Cache line invalidations are needed to keep the 486 microprocessor's cache contents consistent with external memory.

Refer to Section 7.2.8 for further information on cache line invalidations.

5.5 Cache Replacement

When a line needs to be placed in its internal cache the 486 microprocessor first checks to see if there is a non-valid line in the set that can be replaced. If all four lines in the set are valid, a pseudo least-recently-used mechanism is used to determine which line should be replaced.

A valid bit is associated with each line in the cache. When a line needs to be placed in a set, the four

valid bits are checked to see if there is a non-valid line that can be replaced. If a non-valid line is found, that line is marked for replacement.

The four lines in the set are labeled I0, I1, I2, and I3. The order in which the valid bits are checked during an invalidation is I0, I1, I2 and I3. All valid bits are cleared when the processor is reset or when the cache is flushed.

Replacement in the cache is handled by a pseudo least recently used (LRU) mechanism when all four lines in a set are valid. Three bits, B0, B1 and B2, are defined for each of the 128 sets in the cache. These bits are called the LRU bits. The LRU bits are updated for every hit or replace in the cache.

If the most recent access to the set was to I0 or I1, B0 is set to 1. B0 is set to 0 if the most recent access was to I2 or I3. If the most recent access to I0:I1 was to I0, B1 is set to 1, else B1 is set to 0. If the most recent access to I2:I3 was to I2, B2 is set to 1, else B2 is set to 0.

The pseudo LRU mechanism works in the following manner. When a line must be replaced, the cache will first select which of I0:I1 and I2:I3 was least recently used. Then the cache will determine which of the two lines was least recently used and mark it for replacement. This decision tree is shown in Figure 5.2. When the processor is reset or when the cache is flushed all 128 sets of three LRU bits are set to 0.

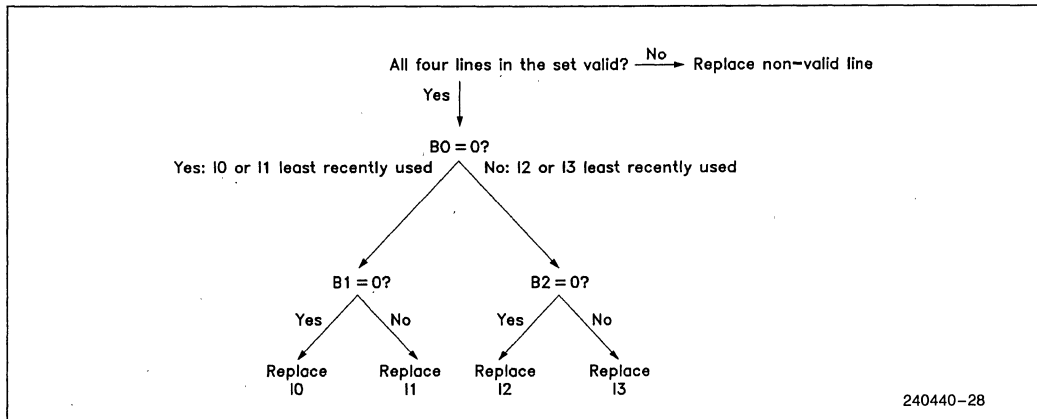


Figure 5.2. On-Chip Cache Replacement Strategy

240440-28

5.6 Page Cacheability

Two bits for cache control, PWT and PCD, are defined in the page table and page directory entries. The state of these bits are driven out on the PWT and PCD pins during memory access cycles.

The PWT bit controls write policy for second level caches used with the 486 microprocessor. Setting PWT=1 defines a write-through policy for the current page while PWT=0 allows the possibility of

write-back. The state of PWT is ignored internally by the 486 microprocessor since the on-chip cache is write through.

The PCD bit controls cacheability on a page by page basis. The PCD bit is internally ANDed with the KEN# signal to control cacheability on a cycle by cycle basis (see Figure 5.3). PCD=0 enables caching while PCD=1 forbids it. Note that cache fills are enabled when PCD=0 AND KEN#=0. This logical AND is implemented physically with a NOR gate.

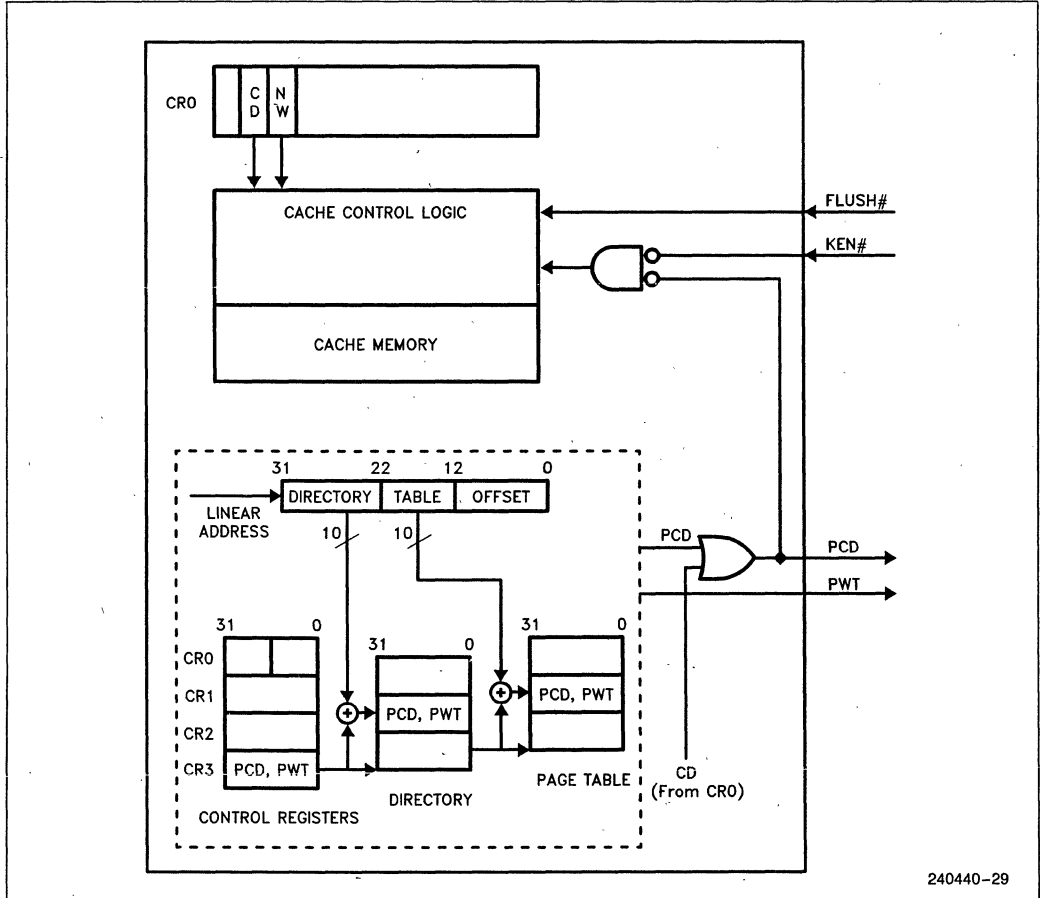


Figure 5.3. Page Cacheability

The state of the PCD bit in the page table entry is driven on the PCD pin when a page in external memory is accessed. The state of the PCD pin informs the external system of the cacheability of the requested information. The external system then returns KEN# telling the 486 microprocessor if the area is cacheable. The 486 microprocessor initiates a cache line fill if PCD and KEN# indicate that the requested information is cacheable.

The PCD bit is masked with the CD (cache disable) bit in control register 0 to determine the state of the PCD pin. If CD=1 the 486 microprocessor forces the PCD pin HIGH. If CD=0 the PCD pin is driven with the value for the page table entry/directory. See Figure 5.3.

The PWT and PCD bits for a bus cycle are obtained from either CR3, the page directory or page table entry. If paging is not enabled, or for cycles that bypass paging, (I/O references, interrupt acknowledge and Halt cycles), the PWT and PCD bits are taken from CR3. These bits are initialized to 0 on reset, but can be set to any value by level 0 software.

When paging is enabled, the bits from the page table entry are cached in the TLB, and are driven any time the page mapped by the TLB entry is referenced. For normal memory cycles, PWT and PCD are taken from the page table entry. During TLB refresh cycles where the page table and directory entries are read, the PWT and PCD bits must be obtained elsewhere. During page table updates the bits are obtained from the page directory. When the page directory is updated the bits are obtained from CR3.

5.7 Cache Flushing

The on-chip cache can be flushed by external hardware or by software instructions. Flushing the cache clears all valid bits for all lines in the cache. The cache is flushed when external hardware asserts the FLUSH# pin.

The instructions INVD and WBINVD cause the on-cache to be flushed. External caches connected to the 486 microprocessor are signalled to flush their contents when these instructions are executed.

WBINVD will cause an external write-back cache to write back dirty lines before flushing its contents. The external cache is signalled using the bus cycle definition pins and the byte enables (refer to Section 6.2.5 for the bus cycle definition pins and Section 7.2.11 for special bus cycles). Refer to the 486 microprocessor programmers reference manual for detailed instruction definitions.

The results of the INVD and WBINVD instructions are identical for the operation of the 486 microprocessor's on-chip cache since the cache is write-through. Note that the INVD and WBINVD instructions are machine dependent. Future members of the 486 microprocessor family may change the definition of this instruction.

5.8 Caching Translation Lookaside Buffer Entries

The 486 microprocessor contains an integrated paging unit with a translation lookaside buffer (TLB). The TLB contains 32 entries. The TLB has been enhanced over the 386 microprocessor's TLB by upgrading the replacement strategy to a pseudo-LRU (least recently used) algorithm. The pseudo-LRU replacement algorithm is the same as that used in the on-chip cache.

The paging TLB operation is automatic whenever paging is enabled. The TLB contains the most recently used page table entries. A page table entry translates the linear address pointing to a particular page to the physical address where the page is stored in memory (refer to Section 4.5, **Paging**).

The paging unit will look up the linear address in the TLB in response to an internal bus request. The corresponding physical address is passed on to the on-chip cache or the external bus (in the event of a cache miss) when the linear address is present in the TLB.

The paging unit will access the page tables in external memory if the linear address is not in the TLB. The required page table entry will be read into the TLB and then the cache or bus cycle for the actual data will take place. The process of reading a new page table entry into the TLB is called a TLB refresh.

A TLB refresh is a two step process. The paging unit must first read the page directory entry which points to the appropriate page table. The page table entry to be stored in the TLB is then read from the page table. Control register 3 (CR3) points to the base of the page directory table.

The 486 microprocessor will allow page directory and page table entries (returned during TLB refreshes) to be stored in the on-chip cache. Setting the PCD bits in CR3 and the page directory entry to 1 will prevent the page directory and page table entries from being stored in the on-chip cache (see Section 5.6, **Page Cacheability**).

6.0 HARDWARE INTERFACE

6.1 Introduction

The 486 microprocessor bus has been designed to be similar to the 386 microprocessor bus whenever possible. Several new features have been added to the 486 microprocessor bus resulting in increased performance and functionality. New features include a 1X clock, a burst bus mechanism for high-speed internal cache fills, a cache line invalidation mechanism, enhanced bus arbitration capabilities, a BSB # bus sizing mechanism and parity support.

The 486 microprocessor is driven by a 1X clock as opposed to a 2X clock in the 386 microprocessor. A 25 MHz 486 microprocessor uses a 25 MHz clock in contrast to a 25 MHz 386 microprocessor which requires a 50 MHz clock. A 1X clock allows simpler system design by cutting in half the clock speed required in the external system.

Like the 386 microprocessor, the 486 microprocessor has separate parallel busses for data and addresses. The bidirectional data bus is 32 bits in width. The address bus consists of two components: 30 address lines (A2–A31) and 4 byte enable lines (BE0#–BE3#). The address bus addresses exter-

nal memory in the same manner as the 386 microprocessor: The address lines form the upper 30 bits of the address and the byte enables select individual bytes within a 4 byte location. The address lines are bidirectional for use in cache line invalidations.

The 486 microprocessor's burst bus mechanism enables high-speed cache fills from external memory. Burst cycles can strobe data into the processor at a rate of one item every clock. Non-burst cycles have a maximum rate of one item every two clocks. Burst cycles are not limited to cache fills: all bus cycles requiring more than a single data cycle can be bursted.

The 486 microprocessor has a bus hold feature similar to that of the 386 microprocessor. During bus hold, the 486 microprocessor relinquishes control of the local bus by floating its address, data and control busses.

The 486 microprocessor has an address hold feature in addition to bus hold. During address hold only the address bus is floated, the data and control busses can remain active. Address hold is used for cache line invalidations.

Ahead is a brief description of the 486 microprocessor input and output signals arranged by functional

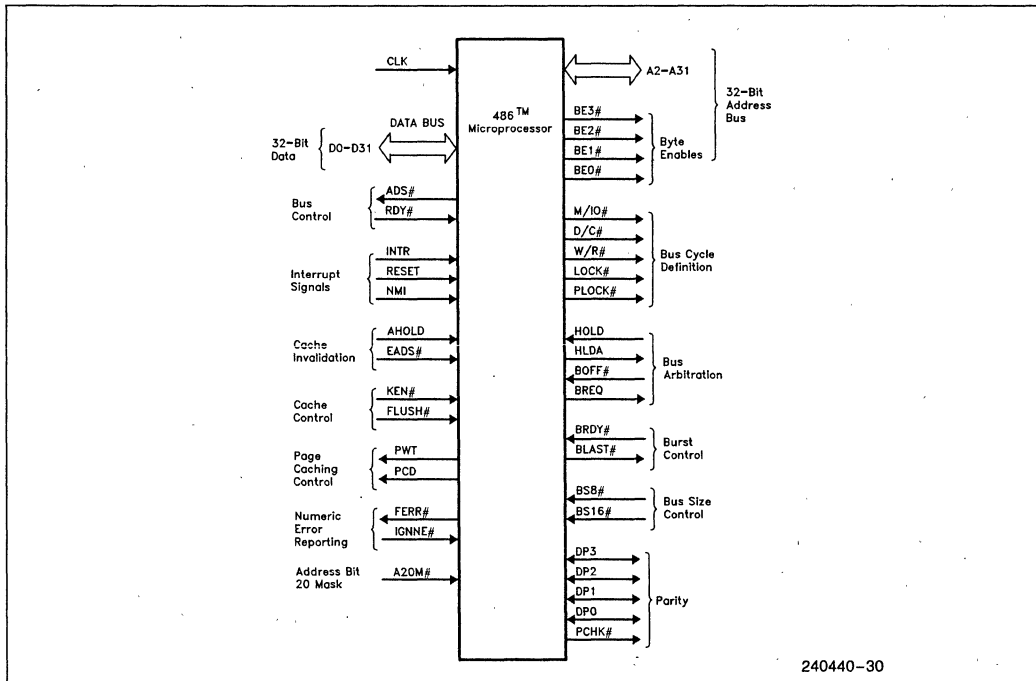


Figure 6.1. Functional Signal Groupings

groups. Before beginning the signal descriptions a few terms need to be defined. The # symbol at the end of a signal name indicates the active, or asserted, state occurs when the signal is at a low voltage. When a # is not present after the signal name, the signal is active at the high voltage level. The term "ready" is used to indicate that the cycle is terminated with RDY# or BRDY#.

Section 6 and 7 will discuss bus cycles and data cycles. A bus cycle is at least two clocks long and begins with ADS# active in the first clock and ready active in the last clock. Data is transferred to or from the 486 microprocessor during a data cycle. A bus cycle contains one or more data cycles.

6.2 Signal Descriptions

6.2.1 CLOCK (CLK)

CLK provides the fundamental timing and the internal operating frequency for the 486 microprocessor. All external timing parameters are specified with respect to the rising edge of CLK.

The 486 microprocessor can operate over a wide frequency range but CLK's frequency cannot change rapidly while RESET is inactive. CLK's frequency must be stable for proper chip operation since a single edge of CLK is used internally to generate two phases. CLK only needs TTL levels for proper operation. Figure 6.2 illustrates the CLK waveform.

6.2.2 Address Bus (A31–A2, BE0#–BE3#)

A31–A2 and BE0#–BE3# form the address bus and provide physical memory and I/O port address-

es. The 486 microprocessor is capable of addressing 4 gigabytes of physical memory space (00000000H through FFFFFFFFH), and 64 Kbytes of I/O address space (00000000H through 0000FFFFH). A31–A2 identify addresses to a 4-byte location. BE0#–BE3# identify which bytes within the 4-byte location are involved in the current transfer.

Addresses are driven back into the 486 microprocessor over A31–A4 during cache line invalidations. The address lines are active HIGH. When used as inputs into the processor, A31–A4 must meet the setup and hold times, t_{22} and t_{23} . A31–A2 are not driven during bus or address hold.

The byte enable outputs, BE0#–BE3#, determine which bytes must be driven valid for read and write cycles to external memory.

- BE3# applies to D24–D31
- BE2# applies to D16–D23
- BE1# applies to D8–D15
- BE0# applies to D0–D7

BE0#–BE3# can be decoded to generate A0, A1 and BHE# signals used in 8- and 16-bit systems (see Table 7.5). BE0#–BE3# are active LOW and are not driven during bus hold.

6.2.3 DATA LINES (D31–D0)

The bidirectional lines, D31–D0, form the data bus for the 486 microprocessor. D0–D7 define the least significant byte and D24–D31 the most significant byte. Data transfers to 8- or 16-bit devices is possible using the data bus sizing feature controlled by the BS8# or BS16# input pins.

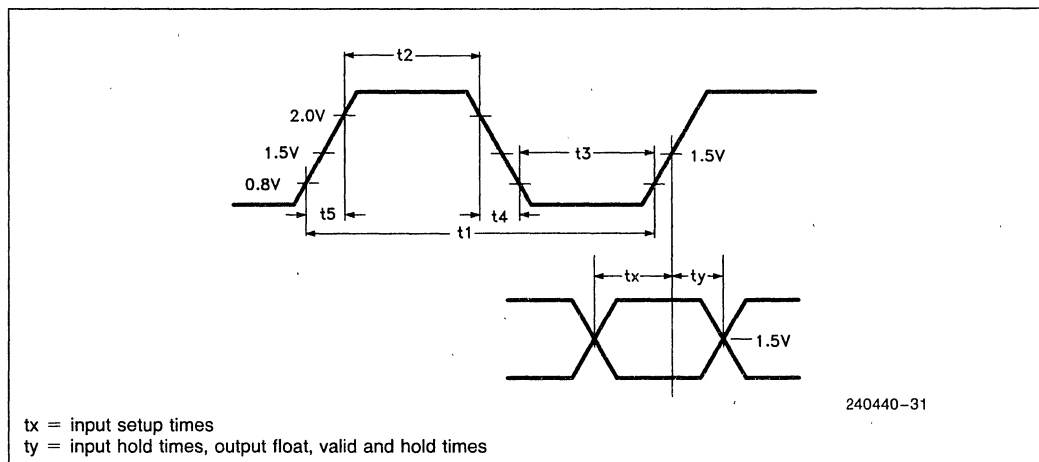


Figure 6.2. CLK waveform

D31–D0 are active HIGH. For reads, D31–D0 must meet the setup and hold times, t_{22} and t_{23} . D31–D0 are not driven during read cycles and bus hold.

6.2.4 PARITY

Data Parity Input/Outputs (DP0–DP3)

DP0–DP3 are the data parity pins for the processor. There is one pin for each byte of the data bus. Even parity is generated or checked by the parity generators/checkers. Even parity means that there are an even number of HIGH inputs on the eight corresponding data bus pins and parity pin.

Data parity is generated on all write data cycles with the same timing as the data driven by the 486 microprocessor. Even parity information must be driven back to the 486 microprocessor on these pins with the same timing as read information to insure that the correct parity check status is indicated by the 486 microprocessor.

The values read on these pins do not affect program execution. It is the responsibility of the system to take appropriate actions if a parity error occurs.

Input signals on DP0–DP3 must meet setup and hold times t_{22} and t_{23} for proper operation.

Parity Status Output (PCHK#)

Parity status is driven on the PCHK# pin, and a parity error is indicated by this pin being LOW. PCHK# is driven the clock after ready for read operations to indicate the parity status for the data sampled at the end of the previous clock. Parity is checked during code reads, memory reads and I/O reads. Parity is not checked during interrupt acknowledge cycles. PCHK# only checks the parity status for enabled bytes as indicated by the byte enable and bus size signals. It is valid only in the clock immediately after read data is returned to the 486 microprocessor. At all other times it is inactive (HIGH). PCHK# is never floated.

Driving PCHK# is the only effect that bad input parity has on the 486 microprocessor. The 486 microprocessor will not vector to a bus error interrupt when bad data parity is returned. In systems that will not employ parity, PCHK# can be ignored. In systems not using parity, DP0–DP3 should be connected to V_{CC} through a pullup resistor.

6.2.5 BUS CYCLE DEFINITION

M/I/O#, D/C#, W/R# Outputs

M/I/O#, D/C# and W/R# are the primary bus cycle definition signals. They are driven valid as the ADS# signal is asserted. M/I/O# distinguishes between memory and I/O cycles, D/C# distinguishes between data and control cycles and W/R# distinguishes between write and read cycles.

Bus cycle definitions as a function of M/I/O#, D/C# and W/R# are given in Table 6.1. Note there is a difference between the 486 microprocessor and 386 microprocessor bus cycle definitions. The halt bus cycle type has been moved to location 001 in the 486 microprocessor from location 101 in the 386 microprocessor. Location 101 is now reserved and will never be generated by the 486 microprocessor.

Table 6.1. AD5# Initiated Bus Cycle Definitions

M/I/O#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Halt/Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

Special bus cycles are discussed in Section 7.2.11.

Bus Lock Output (LOCK#)

LOCK# indicates that the 486 microprocessor is running a read-modify-write cycle where the external bus must not be relinquished between the read and write cycles. Read-modify-write cycles are used to implement memory-based semaphores. Multiple reads or writes can be locked.

When LOCK# is asserted, the current bus cycle is locked and the 486 microprocessor should be allowed exclusive access to the system bus. LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after ready is returned indicating the last locked bus cycle.

The 486 microprocessor will not acknowledge bus hold when LOCK# is asserted (though it will allow an address hold). LOCK# is active LOW and is floated during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN# is returned active. Refer to Section 7.2.6 for a detailed discussion of Locked bus cycles.

Pseudo-Lock Output (PLOCK#)

The pseudo-lock feature allows atomic reads and writes of memory operands greater than 32 bits. These operands require more than one cycle to transfer. The 486 microprocessor asserts PLOCK# during floating point long reads and writes (64 bits), segment table descriptor reads (64 bits) and cache line fills (128 bits).

When PLOCK# is asserted no other master will be given control of the bus between cycles. A bus hold request (HOLD) is not acknowledged during pseudo-locked reads and writes. The 486 microprocessor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether BRDY# or RDY# are returned.

A pseudo-locked transfer is meaningful only if the memory operand is aligned and if its completely contained within a single cache line. A 64-bit floating point number must be aligned to an 8-byte boundary to guarantee an atomic access.

Normally PLOCK# and BLAST# are inverse of each other. However during the first cycle of a 64-bit floating point write, both PLOCK# and BLAST# will be asserted.

Since PLOCK# is a function of the bus size and KEN# inputs, PLOCK# should be sampled only in the clock ready is returned. This pin is active LOW and is not driven during bus hold. Refer to Section 7.2.7 for a detailed discussion of pseudo-locked bus cycles.

6.2.6 BUS CONTROL

The bus control signals allow the processor to indicate when a bus cycle has begun, and allow other system hardware to control burst cycles, data bus width and bus cycle termination.

Address Status Output (ADS#)

The ADS# output indicates that the address and bus cycle definition signals are valid. This signal will go active in the first clock of a bus cycle and go inactive in the second and subsequent clocks of the cycle. ADS# is also inactive when the bus is idle.

ADS# is used by external bus circuitry as the indication that the processor has started a bus cycle. The external circuit must sample the bus cycle definition pins on the next rising edge of the clock after ADS# is driven active.

ADS# is active LOW and is not driven during bus hold.

Non-burst Ready Input (RDY#)

RDY# indicates that the current bus cycle is complete. In response to a read, RDY# indicates that the external system has presented valid data on the data pins. In response to a write request, RDY# indicates that the external system has accepted the 486 microprocessor data. RDY# is ignored when the bus is idle and at the end of the first clock of the bus cycle. Since RDY# is sampled during address hold, data can be returned to the processor when AHOLD is active.

RDY# is active LOW, and is not provided with an internal pullup resistor. This input must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

6.2.7 BURST CONTROL

Burst Ready Input (BRDY#)

BRDY# performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted the 486 microprocessor data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle.

During a burst cycle, BRDY# will be sampled each clock, and if active, the data presented on the data bus pins will be strobed into the 486 microprocessor. ADS# is negated during the second through last data cycles in the burst, but address lines A2–A3 and byte enables will change to reflect the next data item expected by the 486 microprocessor.

If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely aborted. An additional complete bus cycle will be initiated after an aborted burst cycle if the cache line fill was not complete. BRDY# is treated as a normal ready for the last data cycle in a burst transfer or for non-burstable cycles. Refer to Section 7.2.2 for burst cycle timing.

BRDY# is active LOW and is provided with a small internal pullup resistor. BRDY# must satisfy the setup and hold times t_{16} and t_{17} .

Burst Last Output (BLAST#)

BLAST# indicates that the next time BRDY# is returned it will be treated as a normal RDY#, terminating the line fill or other multiple-data-cycle transfer. BLAST# is active for all bus cycles regardless of whether they are cacheable or not. This pin is active LOW and is not driven during bus hold.

6.2.8 INTERRUPT SIGNALS (RESET, INTR, NMI)

The interrupt signals can interrupt or suspend execution of the processor's current instruction stream.

Reset Input (RESET)

RESET forces the 486 microprocessor to begin execution at a known state. V_{CC} and CLK must reach their proper DC and AC specifications for at least 1 ms before the 486 microprocessor begins instruction execution. The RESET pin should remain active during this time to ensure proper 486 microprocessor operation. The testability operating modes are programmed by the falling (inactive going) edge of RESET. (Refer to Section 8.0 for a description of the test modes during reset.)

Maskable Interrupt Request Input (INTR)

INTR indicates that an external interrupt has been generated. Interrupt processing is initiated if the IF flag is active in the EFLAGS register.

The 486 microprocessor will generate two locked interrupt acknowledge bus cycles in response to asserting the INTR pin. An 8-bit interrupt number will be latched from an external interrupt controller at the end of the second interrupt acknowledge cycle. INTR must remain active until the interrupt acknowledges have been performed to assure program interruption. Refer to Section 7.2.10 for a detailed discussion of interrupt acknowledge cycles.

The INTR pin is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but the INTR setup and hold times, t_{20} and t_{21} , must be met to assure recognition on any specific clock.

Non-maskable Interrupt Request Input (NMI)

NMI is the non-maskable interrupt request signal. Asserting NMI causes an interrupt with an internally supplied vector value of 2. External interrupt acknowledge cycles are not generated since the NMI interrupt vector is internally generated. When NMI processing begins, the NMI signal will be masked internally until the IRET instruction is executed.

NMI is rising edge sensitive after internal synchronization. NMI must be held LOW for at least four CLK periods before this rising edge for proper operation. NMI is not provided with an internal pulldown resistor. NMI is asynchronous but setup and hold times, t_{20} and t_{21} must be met to assure recognition on any specific clock.

6.2.9 BUS ARBITRATION SIGNALS

This section describes the mechanism by which the processor relinquishes control of its local bus when requested by another bus master.

Bus Request Output (BREQ)

The 486 asserts BREQ whenever a bus cycle is pending internally. Thus, BREQ is always asserted in the first clock of a bus cycle, along with $ADS\#$. Furthermore, if the 486 is currently not driving the bus (due to HOLD, AHOLD, or BOFF#), BREQ is asserted in the same clock that $ADS\#$ would have been asserted if the processor were driving the bus. After the first clock of the bus cycle, BREQ may change state. It will be asserted if additional cycles are necessary to complete a transfer (via $BS8\#$, $BS16\#$, $KEN\#$), or if more cycles are pending internally. However, if no additional cycles are necessary to complete the current transfer, BREQ can be negated before ready comes back for the current cycle. External logic can use the BREQ signal to arbitrate among multiple processors. This pin is driven regardless of the state of bus hold or address hold. BREQ is active HIGH and is never floated.

Bus Hold Request Input (HOLD)

HOLD allows another bus master complete control of the 486 microprocessor bus. The 486 microprocessor will respond to an active HOLD signal by asserting HLDA and placing most of its output and input/output pins in a high impedance state (floated) after completing its current bus cycle, burst cycle, or sequence of locked cycles. The BREQ, HLDA, PCHK# and FERR# pins are not floated during bus hold. The 486 microprocessor will maintain its bus in this state until the HOLD is deasserted. Refer to Section 7.2.9 for timing diagrams for a bus hold cycle.

Unlike the 386 microprocessor, the 486 microprocessor will recognize HOLD during reset. Pullup resistors are not provided for the outputs that are floated in response to HOLD. HOLD is active HIGH and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper chip operation.

Bus Hold Acknowledge Output (HLDA)

HLDA indicates that the 486 microprocessor has given the bus to another local bus master. HLDA goes active in response to a hold request presented on the HOLD pin. HLDA is driven active in the same clock that the 486 microprocessor floats its bus.

HLDA will be driven inactive when leaving bus hold and the 486 microprocessor will resume driving the bus. The 486 microprocessor will not cease internal activity during bus hold since the internal cache will satisfy the majority of bus requests. HLDA is active HIGH and remains driven during bus hold.

Backoff Input (BOFF#)

Asserting the BOFF# input forces the 486 microprocessor to release control of its bus in the next clock. The pins floated are exactly the same as in response to HOLD. The response to BOFF# differs from the response to HOLD in two ways: First, the bus is floated immediately in response to BOFF# while the 486 completes the current bus cycle before floating its bus in response to HOLD. Second the 486 does not assert HLDA in response to BOFF#.

The processor remains in bus hold until BOFF# is negated. Upon negation, the 486 microprocessor restarts the bus cycle aborted when BOFF# was asserted. To the internal execution engine the effect of BOFF# is the same as inserting a few wait states to the original cycle. Refer to Section 7.2.12 for a description of bus cycle restart.

Any data returned to the processor while BOFF# is asserted is ignored. BOFF# has higher priority than RDY# or BRDY#. If both BOFF# and ready are returned in the same clock, BOFF# takes effect. If BOFF# is asserted while the bus is idle, the 486 microprocessor will float its bus in the next clock. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper chip operation.

6.2.10 CACHE INVALIDATION

The AHOLD and EADS# inputs are used during cache invalidation cycles. AHOLD conditions the 486 microprocessors address lines, A4–A31, to accept an address input. EADS# indicates that an external address is actually valid on the address inputs. Activating EADS# will cause the 486 microprocessor to read the external address bus and perform an internal cache invalidation cycle to the address indicated. Refer to Section 7.2.8 for cache invalidation cycle timing.

Address Hold Request Input (AHOLD)

AHOLD is the address hold request. It allows another bus master access to the 486 microprocessor address bus for performing an internal cache invali-

dation cycle. Asserting AHOLD will force the 486 microprocessor to stop driving its address bus in the next clock. While AHOLD is active only the address bus will be floated, the remainder of the bus can remain active. For example, data can be returned for a previously specified bus cycle when AHOLD is active. The 486 microprocessor will not initiate another bus cycle during address hold. Since the 486 microprocessor floats its bus immediately in response to AHOLD, an address hold acknowledge is not required. If AHOLD is asserted while a bus cycle is in progress, and no readies are returned during the time AHOLD is asserted, the 486 will redrive the same address (that it originally sent out) once AHOLD is negated.

AHOLD is recognized during reset. Since the entire cache is invalidated by reset, any invalidation cycles run during reset will be unnecessary. AHOLD is active HIGH and is provided with a small internal pull-down resistor. It must satisfy the setup and hold times t_{18} and t_{19} for proper chip operation. This pin determines whether or not the built in self test features of the 486 microprocessor will be exercised on assertion of RESET.

External Address Valid Input (EADS#)

EADS# indicates that a valid external address has been driven onto the 486 address pins. This address will be used to perform an internal cache invalidation cycle. The external address will be checked with the current cache contents. If the address specified matches any areas in the cache, that area will immediately be invalidated.

An invalidation cycle may be run by asserting EADS# regardless of the state of AHOLD, HOLD and BOFF#. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy the setup and hold times t_{12} and t_{13} for proper chip operation.

6.2.11 CACHE CONTROL

Cache Enable Input (KEN#)

KEN# is the cache enable pin. KEN# is used to determine whether the data being returned by the current cycle is cacheable. When KEN# is active and the 486 microprocessor generates a cycle that can be cached (most any memory read cycle), the cycle will be transformed into a cache line fill cycle.

A cache line is 16 bytes long. During the first cycle of a cache line fill the byte-enable pins should be ignored and data should be returned as if all four byte

enables were asserted. The 486 microprocessor will run between 4 and 16 contiguous bus cycles to fill the line depending on the bus data width selected by BS8# and BS16#. Refer to Section 7.2.3 for a description of cache line fill cycles.

The KEN# input is active LOW and is provided with a small internal pullup resistor. It must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

Cache Flush Input (FLUSH#)

The FLUSH# input forces the 486 microprocessor to flush its entire internal cache. FLUSH# is active LOW and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition on any specific clock.

FLUSH# also determines whether or not the tristate test mode of the 486 microprocessor will be invoked on assertion of RESET.

6.2.12 PAGE CACHEABILITY (PWT, PCD)

The PWT and PCD output signals correspond to two user attribute bits in the page table entry. When paging is enabled, PWT and PCD correspond to bits 3 and 4 of the page table entry respectively. When paging is disabled, or for cycles that are not paged when paging is enabled (for example I/O cycles) PWT and PCD correspond to bits 3 and 4 in control register 3.

PCD is masked by the CD (cache disable) bit in control register 0 (CR0). When CD=1 (cache line fills disabled) the 486 microprocessor forces PCD HIGH. When CD=0, PCD is driven with the value of the page table entry/directory.

The purpose of PCD is to provide a cacheable/non-cacheable indication on a page by page basis. The 486 will not perform a cache fill to any page in which bit 4 of the page table entry is set. PWT corresponds to the write-back bit and can be used by an external cache to provide this functionality. Refer to Sections 4.5.4 and 5.6 for a discussion of non-cacheable pages.

PCD and PWT have the same timing as the cycle definition pins (M/IO#, D/C#, W/R#). PCD and PWT are active HIGH and are not driven during bus hold.

6.2.13 NUMERIC ERROR REPORTING (FERR#, IGNNE#)

To allow PC-type floating point error reporting, the 486 microprocessor provides two pins, FERR# and IGNNE#.

Floating Point Error Output (FERR#)

The 486 microprocessor asserts FERR# whenever an unmasked floating point error is encountered. FERR# is similar to the ERROR# pin on the 387 math coprocessor. FERR# can be used by external logic for PC-type floating point error reporting in 486 microprocessor systems. FERR# is active LOW, and is not floated during bus hold.

Ignore Numeric Error Input (IGNNE#)

The 486 microprocessor will ignore a numeric error and continue executing non-control floating point instructions when IGNNE# is asserted. When deasserted, the 486 microprocessor will freeze on a non-control floating point instruction if a previous instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set.

The IGNNE# input is active LOW and is provided with a small internal pullup resistor. This input is asynchronous, but must meet setup and hold times t_{20} and t_{21} to insure recognition on any specific clock.

6.2.14 BUS SIZE CONTROL (BS16#, BS8#)

The BS16# and BS8# inputs allow external 16- and 8-bit busses to be supported with a small number of external components. The 486 CPU samples these pins every clock. The value sampled in the clock before ready determines the bus size. When asserting BS16# or BS8# only 16 or 8 bits of the data bus need be valid. If both BS16# and BS8# are asserted, an 8-bit bus width is selected.

When BS16# or BS8# are asserted the 486 microprocessor will convert a larger data request to the appropriate number of smaller transfers. The byte enables will also be modified appropriately for the bus size selected.

BS16# and BS8# are active LOW and are provided with small internal pullup resistors. BS16# and BS8# must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

6.2.15 ADDRESS BIT 20 MASK (A20M#)

Asserting the A20M# input causes the 486 microprocessor to mask physical address bit 20 before performing a lookup in the internal cache and before driving a memory cycle to the outside world. When A20M# is asserted, the 486 microprocessor emulates the 1 Mbyte address wraparound that occurs on the 8086. A20M# is active LOW and must be asserted only when the processor is in real mode. A20M# is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For correct operation of the chip, A20M# should be sampled high at the falling edge of RE-SET.

6.3 Write Buffers

The 486 microprocessor contains four write buffers to enhance the performance of consecutive writes to memory. The buffers can be filled at a rate of one write per clock until all four buffers are filled.

When all four buffers are empty and the bus is idle, a write request will propagate directly to the external bus bypassing the write buffers. If the bus is not available at the time the write is generated internally, the write will be placed in the write buffers and propagate to the bus as soon as the bus becomes available. The write is stored in the on-chip cache immediately if the write is a cache hit.

Writes will be driven onto the external bus in the same order in which they are received by the write buffers. Under certain conditions a memory read will go onto the external bus before the memory writes pending in the buffer even though the writes occurred earlier in the program execution.

A memory read will only be reordered in front of all writes in the buffers under the following conditions: If all writes pending in the buffers are cache hits and the read is a cache miss. Under these conditions the 486 microprocessor will not read from an external memory location that needs to be updated by one of the pending writes.

Reordering of a read with the writes pending in the buffers can only occur once before all the buffers are emptied. The problem with reordering more than one write is illustrated with the following example. A write to external memory location M is pending in the write buffers. This write was a cache hit to location C in the on-chip cache. A read is reordered ahead of the write to location M. The data from this read replaces the data in location C in the on-chip cache. Before the pending write can update location M the processor generates a read to location M. This read is a cache miss since the previous read replaced the

information in on-chip location C. If the 486 microprocessor reordered this second read ahead of the pending write to location M, the processor would read stale data.

6.3.1 WRITE BUFFERS AND I/O CYCLES

Input/Output (I/O) cycles must be handled in a different manner by the write buffers.

I/O reads are never reordered in front of buffered memory writes. This insures that the 486 microprocessor will update all memory locations before reading status from an I/O device.

The 486 microprocessor never buffers single I/O writes. When processing an OUT instruction, internal execution stops until the I/O write actually completes on the external bus. This allows time for the external system to drive an invalidate into the 486 microprocessor or to mask interrupts before the processor progresses to the instruction following OUT. Repeated OUTS instructions will be buffered.

I/O device recovery time must be handled slightly differently by the 486 microprocessor than with the 386 microprocessor. I/O device back-to-back write recovery times could be guaranteed by the 386 microprocessor by inserting a jump to the next instruction in the code that writes to the device. The jump forces the 386 microprocessor to generate a prefetch bus cycle which can't begin until the I/O write completes.

Inserting a jump to the next write will not work with the 486 microprocessor because the prefetch could be satisfied by the on-chip cache. A read cycle must be explicitly generated to a non-cacheable location in memory to guarantee that a read bus cycle is performed. This read will not be allowed to proceed to the bus until after the I/O write has completed because I/O writes are not buffered. The I/O device will have time to recover to accept another write during the read cycle.

6.3.2 WRITE BUFFERS IMPLICATIONS ON LOCKED BUS CYCLES

Locked bus cycles are used for read-modify-write accesses to memory. During a read-modify-write access, a memory base variable is read, modified and then written back to the same memory location. It is important that no other bus cycles, generated by other bus masters or by the 486 microprocessor itself, be allowed on the external bus between the read and write portion of the locked sequence.

During a locked read cycle the 486 microprocessor will always access external memory, it will never

look for the location in the on-chip cache. All data pending in the 486 microprocessor's write buffers will be written to memory before a locked cycle is allowed to proceed to the external bus.

The 486 microprocessor will assert the LOCK# pin after the write buffers are emptied during a locked bus cycle. With the LOCK# pin asserted, the microprocessor will read the data, operate on the data and place the results in a write buffer. The contents of the write buffer will then be written to external memory. LOCK# will become inactive after the write part of the locked cycle.

6.4 Interrupt and Non-Maskable Interrupt Interface

The 486 microprocessor provides two asynchronous interrupt inputs, INTR (interrupt request) and NMI (non-maskable interrupt input). This section describes the hardware interface between the instruction execution unit and the pins. For a description of the algorithmic response to interrupts refer to Section 2.7. For interrupt timings refer to Section 7.2.10.

6.4.1 INTERRUPT LOGIC

The 486 microprocessor contains a two-clock synchronizer on the interrupt line. An interrupt request will reach the internal instruction execution unit two clocks after the INTR pin is asserted, if proper setup is provided to the first stage of the synchronizer. There is no special logic in the interrupt path other than the synchronizer. The INTR signal is level sensitive and must remain active for the instruction execution unit to recognize it. The interrupt will not be serviced by the 486 microprocessor if the INTR signal does not remain active.

The instruction execution unit will look at the state of the synchronized interrupt signal at specific clocks during the execution of instructions (if interrupts are enabled). These specific clocks are at instruction boundaries, or iteration boundaries in the case of string move instructions. Interrupts will only be accepted at these boundaries.

An interrupt must be presented to the 486 microprocessor INTR pin three clocks before the end of an instruction for the interrupt to be acknowledged. Presenting the interrupt 3 clocks before the end of an instruction allows the interrupt to pass through the two clock synchronizer leaving one clock to prevent the initiation of the next sequential instruction and to begin interrupt service. If the interrupt is not received in time to prevent the next instruction, it will be accepted at the end of next instruction, assuming INTR is still held active. The interrupt service microcode will start after two dead clocks.

The longest latency between when an interrupt request is presented on the INTR pin and when the interrupt service begins is: longest instruction used + the two clocks for synchronization + one clock required to vector into the interrupt service microcode.

6.4.2 NMI LOGIC

The NMI pin has a synchronizer like that used on the INTR line. Other than the synchronizer, the NMI logic is different from that of the maskable interrupt.

NMI is edge triggered as opposed to the level triggered INTR signal. The rising edge of the NMI signal is used to generate the interrupt request. The NMI input need not remain active until the interrupt is actually serviced. The NMI pin only needs to remain active for a single clock if the required setup and hold times are met. NMI will operate properly if it is held active for an arbitrary number of clocks.

The NMI input must be held inactive for at least four clocks after it is asserted to reset the edge triggered logic. A subsequent NMI may not be generated if the NMI is not held inactive for at least two clocks after being asserted.

The NMI input is internally masked whenever the NMI routine is entered. The NMI input will remain masked until an IRET (return from interrupt) instruction is executed. Masking the NMI signal prevents recursive NMI calls. If another NMI occurs while the NMI is masked off, the pending NMI will be executed after the current NMI is done. Only one NMI can be pending while NMI is masked.

6.5 Reset and Initialization

The 486 microprocessor has a built in self test (BIST) that can be run during reset. The BIST is invoked if the AHOLD pin is asserted on the falling edge of RESET. Refer to Section 8.0 for information on 486 microprocessor testability.

The 486 microprocessor registers have the values shown in Table 6.2 after RESET is performed. The EAX register contains information on the success or failure of the BIST if the self test is executed. The DX register always contains a component identifier at the conclusion of RESET. The upper byte of DX (DH) will contain 04 and the lower byte (DL) will contain a stepping identifier. The floating point registers are initialized as if the FINIT/FNINIT (initialize processor) instruction was executed if the BIST was performed. If the BIST is not executed, the floating point registers are unchanged.

Table 6.2. Register Values after Reset

Register	Initial Value (BIST)	Initial Value (No Bist)
EAX	Zero (Pass)	Undefined
ECX	Undefined	Undefined
EDX	0400 + Revision ID	0400 + Revision ID
EBX	Undefined	Undefined
ESP	Undefined	Undefined
EBP	Undefined	Undefined
ESI	Undefined	Undefined
EDI	Undefined	Undefined
EFLAGS	0000002h	0000002h
EIP	0FFF0h	0FFF0h
ES	0000h	0000h
CS	F000h*	F000h*
SS	0000h	0000h
DS	0000h	0000h
FS	0000h	0000h
GS	0000h	0000h
IDTR	Base = 0, Limit = 3FFh	Base = 0, Limit = 3FFh
CR0	60000000h	60000000h
DR7	00000000h	00000000h
CW	037Fh	Unchanged
SW	0000h	Unchanged
TW	FFFFh	Unchanged
FIP	00000000h	Unchanged
FEA	00000000h	Unchanged
FCS	0000h	Unchanged
FDS	0000h	Unchanged
FOP	000h	Unchanged
FSTACK	Undefined	Unchanged

The 486 microprocessor will start executing instructions at location FFFFFFF0H after RESET. When the first InterSegment Jump or Call is executed, address lines A20–A31 will drop LOW for CS-relative memory cycles, and the 486 microprocessor will only execute instructions in the lower one Mbyte of physical memory. This allows the system designer to use a ROM at the top of physical memory to initialize the system and take care of RESETs.

RESET forces the 486 microprocessor to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active.

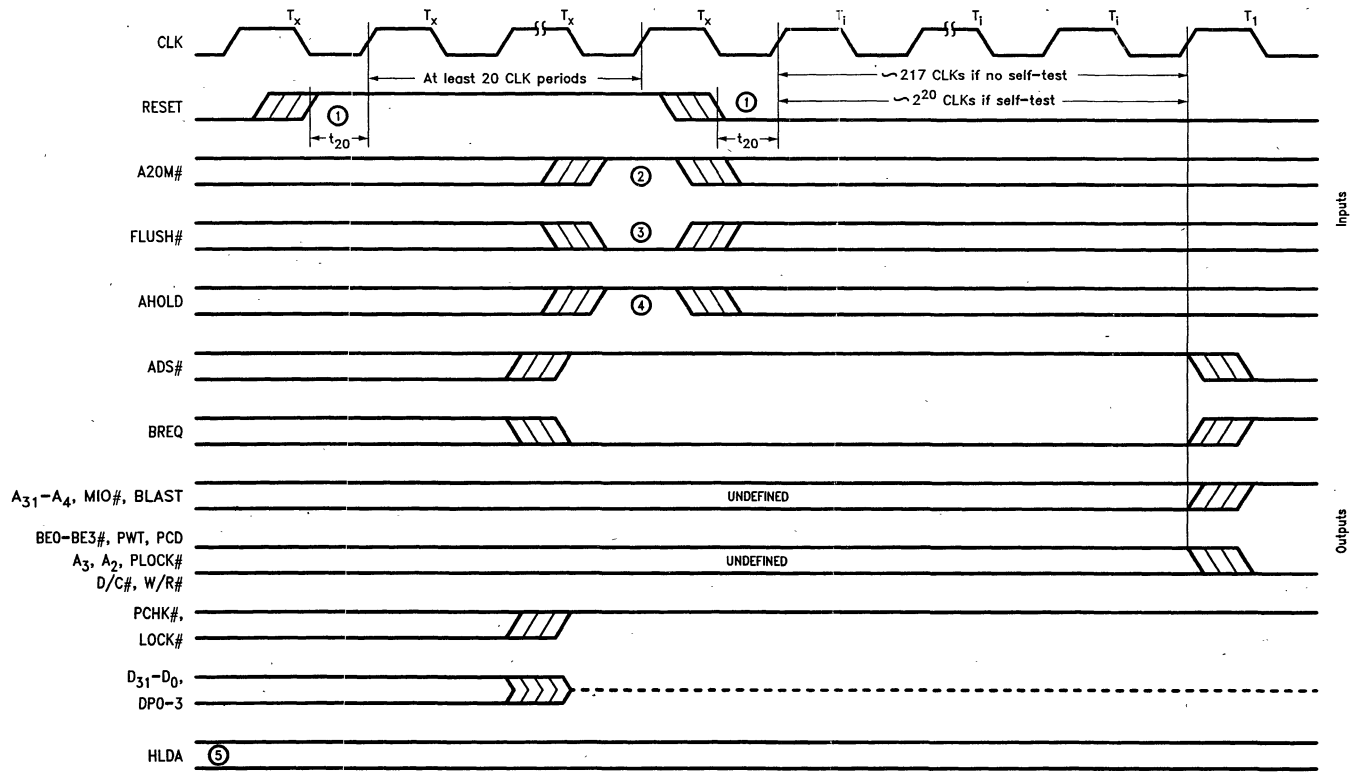
All entries in the cache are invalidated by RESET.

6.5.1 PIN STATE DURING RESET

The 486 microprocessor recognizes and can respond to HOLD, AHOLD, and BOFF# requests re-

gardless of the state of RESET. Thus, even though the processor is in reset, it can still float its bus in response to any of these requests.

While in reset, the 486 microprocessor bus is in the state shown in Figure 6.3 if the HOLD, AHOLD and BOFF# requests are inactive. Note that the address (A31–A2, BE3#–BE0#) and cycle definition (M/IO#, D/C#, W/R#) pins are undefined from the time reset is asserted up to the start of the first bus cycle. All undefined pins (except FERR#) assume known values at the beginning of the first bus cycle. The first bus cycle is always a code fetch to address FFFFFFF0H. FERR# reflects the state of the ES (error summary status) bit in the floating point unit status word. The ES bit is initialized whenever the floating point unit state is initialized.



240440-32

NOTES:

1. RESET is an asynchronous input. t_{20} must be met only to guarantee recognition on a specific clock edge.
2. High on this clock edge for correct operation of the part.
3. Low on this clock edge if tri-state output test mode is to be entered.
4. High on this clock edge to initiate self-test.
5. Hold is recognized normally during RESET.

 Figure 6.3. Pin States during RESET
 4-86

7.0 BUS OPERATION

7.1 Data Transfer Mechanism

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte, word and dword lengths may be transferred without restrictions on physical address alignment. Data may be accessed at any byte boundary but two or three cycles may be required for unaligned data transfers. See Section 7.1.3 Dynamic Bus Sizing and 7.1.6 Operand Alignment.

The 486 microprocessor address signals are split into two components. High-order address bits are provided by the address lines, A2–A31. The byte enables, BE0#–BE3#, form the low-order address and provide linear selects for the four bytes of the 32-bit address bus.

The byte enable outputs are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 7.1. Byte enable patterns which have a negated byte enable separating two or three asserted byte enables will never occur (see Table 7.5). All other byte enable patterns are possible.

Table 7.1. Byte Enables and Associated Data and Operand Bytes

Byte Enable Signal	Associated Data Bus Signals
BE0#	D0–D7 (byte 0—least significant)
BE1#	D8–D15 (byte 1)
BE2#	D16–D23 (byte 2)
BE3#	D24–D31 (byte 3—most significant)

Address bits A0 and A1 of the physical operand's base address can be created when necessary. Use of the byte enables to create A0 and A1 is shown in Table 7.2. The byte enables can also be decoded to generate BLE# (byte low enable) and BHE# (byte high enable). These signals are needed to address 16-bit memory systems (see Section 7.1.4 Interfacing with 8- and 16-bit memories).

Table 7.2. Generating A0–A31 from BE0#–BE3# and A2–A31

486™ CPU Address Signals							
A31	A2			BE3#	BE2#	BE1#	BE0#
Physical Base Address							
A31	A2	A1	A0			
A31	A2	0	0	X	X	Low
A31	A2	0	1	X	X	Low High
A31	A2	1	0	X	Low	High
A31	A2	1	1	Low	High	High

7.1.1 MEMORY AND I/O SPACES

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. Physical memory addresses range from 00000000H to FFFFFFFFH (4 gigabytes). I/O addresses range from 00000000H to 0000FFFFH (64 Kbytes) for programmed I/O. See Figure 7.1.

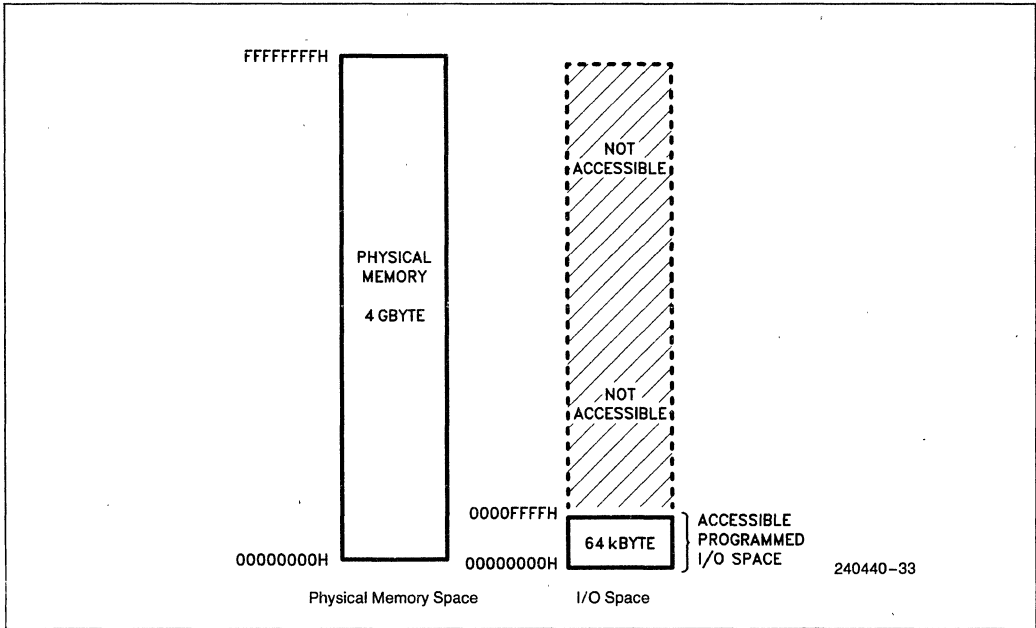


Figure 7.1. Physical Memory and I/O Spaces

7.1.2 MEMORY AND I/O SPACE ORGANIZATION

The 486 microprocessor datapath to memory and input/output (I/O) spaces can be 32-, 16- or 8-bits wide. The byte enable signals, BE0#–BE3#, allow byte granularity when addressing any memory or I/O structure whether 8, 16 or 32 bits wide.

The 486 microprocessor includes bus control pins, BS16# and BS8#, which allow direct connection to 16- and 8-bit memories and I/O devices. Cycles to 32-, 16- and 8-bit may occur in any sequence, since the BS8# and BS16# signals are sampled during each bus cycle.

32-bit wide memory and I/O spaces are organized as arrays of physical 4-byte words. Each memory or I/O 4-byte word has four individually addressable bytes at consecutive byte addresses (see Figure 7.2). The lowest addressed byte is associated with data signals D0–D7; the highest-addressed byte with D24–D31. Physical 4-byte words begin at addresses divisible by four.

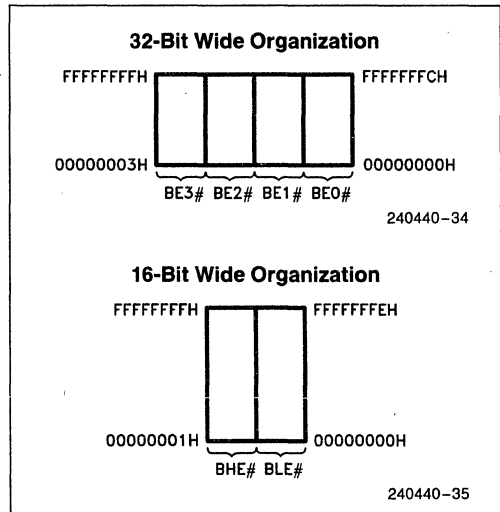


Figure 7.2. Physical Memory and I/O Space Organization

16-bit memories are organized as arrays of physical 2-byte words. Physical 2-byte words begin at addresses divisible by two. The byte enables BE0#–BE3#, must be decoded to A1, BLE# and BHE# to address 16-bit memories (see Section 7.1.4).

To address 8-bit memories, the two low order address bits A0 and A1, must be decoded from BE0#–BE3#. The same logic can be used for 8- and 16-bit memories since the decoding logic for BLE# and A0 are the same (see Section 7.1.4).

7.1.3 DYNAMIC DATA BUS SIZING

Dynamic data bus sizing is a feature allowing processor connection to 32-, 16- or 8-bit buses for memory or I/O. A processor may connect to all three bus sizes. Transfers to or from 32-, 16- or 8-bit devices are supported by dynamically determining the bus width during each bus cycle. Address decoding circuitry may assert BS16# for 16-bit devices, or BS8# for 8-bit devices during each bus cycle. BS8# and BS16# must be negated when addressing 32-bit devices. An 8-bit bus width is selected if both BS16# and BS8# are asserted.

BS16# and BS8# force the 486 microprocessor to run additional bus cycles to complete requests larger than 16- or 8 bits. A 32-bit transfer will be converted into two 16-bit transfers (or 3 transfers if the data is misaligned) when BS16# is asserted. Asserting BS8# will convert a 32-bit transfer into four 8-bit transfers.

Extra cycles forced by BS16# or BS8# should be viewed as independent bus cycles. BS16# or BS8# must be driven active during each of the extra cycles unless the addressed device has the ability to change the number of bytes it can return between cycles.

The 486 microprocessor will drive the byte enables appropriately during extra cycles forced by BS8# and BS16#. A2–A31 will not change if accesses are to a 32-bit aligned area. Table 7.3 shows the set of byte enables that will be generated on the next cycle for each of the valid possibilities of the byte enables on the current cycle.

The dynamic bus sizing feature of the 486 microprocessor is significantly different than that of the 386 microprocessor. Unlike the 386 microprocessor, the 486 microprocessor requires that data bytes be driven on the addressed data pins. The simplest example of this function is a 32-bit aligned, BS16# read. When the 486 microprocessor reads the two high order bytes, they must be driven on the data bus pins D16–D31. The 486 microprocessor expects the two low order bytes on D0–D15. The 386 microprocessor expects both the high and low order bytes on D0–D15. The 386 microprocessor always reads or writes data on the lower 16 bits of the data bus when BS16# is asserted.

The external system must contain buffers to enable the 486 microprocessor to read and write data on the appropriate data bus pins. Table 7.4 shows the data bus lines where the 486 microprocessor expects data to be returned for each valid combination of byte enables and bus sizing options.

Valid data will only be driven onto data bus pins corresponding to active byte enables during write cycles. Other pins in the data bus may be driven but they will not contain valid data. Unlike the 386 microprocessor, the 486 microprocessor will not duplicate write data onto parts of the data bus for which the corresponding byte enable is negated.

Table 7.3. Next Byte Enable Values for BSn# Cycles

Current				Next with BS8#				Next with BS16#			
BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#
1	1	1	0	n	n	n	n	n	n	n	n
1	1	0	0	1	1	0	1	n	n	n	n
1	0	0	0	1	0	0	1	1	0	1	1
0	0	0	0	0	0	0	1	0	0	1	1
1	1	0	1	n	n	n	n	n	n	n	n
1	0	0	1	1	0	1	1	1	0	1	1
0	0	0	1	0	0	1	1	0	0	1	1
1	0	1	1	n	n	n	n	n	n	n	n
0	0	1	1	0	1	1	1	n	n	n	n
0	1	1	1	n	n	n	n	n	n	n	n

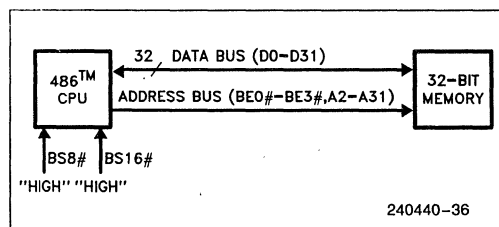
"n" means that another bus cycle will not be required to satisfy the request.

Table 7.4. Data Pins Read with Different Bus Sizes

BE3#	BE2#	BE1#	BE0#	w/o BS8# /BS16#	w BS8#	W BS16#
1	1	1	0	D7-D0	D7-D0	D7-D0
1	1	0	0	D15-D0	D7-D0	D15-D0
1	0	0	0	D23-D0	D7-D0	D15-D0
0	0	0	0	D31-D0	D7-D0	D15-D0
1	1	0	1	D15-D8	D15-D8	D15-D8
1	0	0	1	D23-D8	D15-D8	D15-D8
0	0	0	1	D31-D8	D15-D8	D15-D8
1	0	1	1	D23-D16	D23-D16	D23-D16
0	0	1	1	D31-D16	D23-D16	D31-D16
0	1	1	1	D31-D24	D31-D24	D31-D24

7.1.4 INTERFACING WITH 8-, 16- AND 32-BIT MEMORIES

In 32-bit physical memories such as Figure 7.3, each 4-byte word begins at a byte address that is a multiple of four. A2-A31 are used as a 4-byte word select. BE0#-BE3# select individual bytes within the 4-byte word. BS8# and BS16# are negated for all bus cycles involving the 32-bit array.


Figure 7.3. i486™ Microprocessor with 32-Bit Memory

16- and 8-bit memories require external byte swapping logic for routing data to the appropriate data lines and logic for generating BHE#, BLE# and A1. In systems where mixed memory widths are used, extra address decoding logic is necessary to assert BS16# or BS8#.

Figure 7.4 shows the 486 microprocessor address bus interface to 32-, 16- and 8-bit memories. To address 16-bit memories the byte enables must be decoded to produce A1, BHE# and BLE# and A1. For 8-bit wide memories the byte enables must be decoded to produce A0 and A1. The same byte select logic can be used in 16- and 8-bit systems since BLE# is exactly the same as A0 (see Table 7.5).

BE0#-BE3# can be decoded as shown in Table 7.5 to generate A1, BHE# and BLE#. The byte select logic necessary to generate BHE# and BLE# is shown in Figure 7.5.

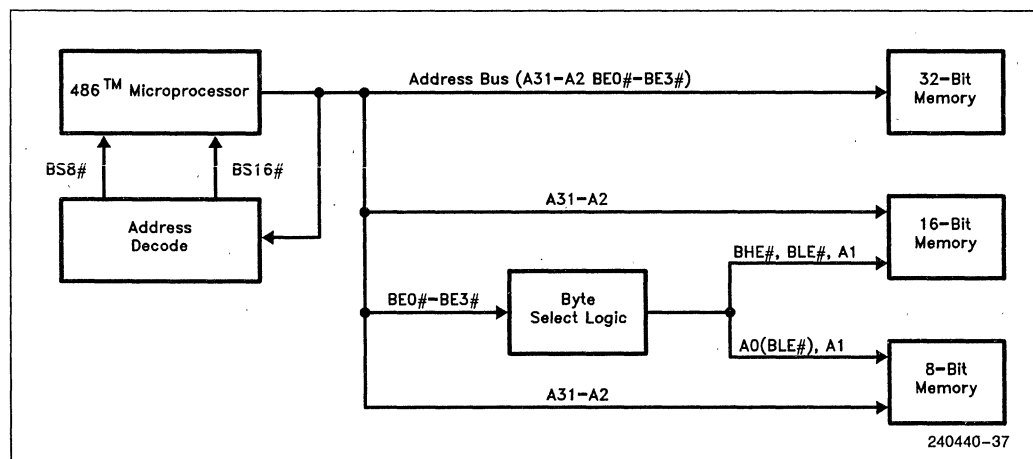

Figure 7.4. Addressing 16- and 8-Bit Memories

Table 7.5. Generating A1, BHE # and BLE # for Addressing 16-Bit Devices

i486™ CPU Signals				8, 16-Bit Bus Signals			Comments
BE3 #	BE2 #	BE1 #	BE0 #	A1	BHE #	BLE # (A0)	
H*	H*	H*	H*	x	x	x	x—no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	x—not contiguous bytes
H*	L*	H*	L*	x	x	x	
H	L	L	H	L	L	H	
L	H	H	H	H	L	H	
L*	H*	H*	L*	x	x	x	x—not contiguous bytes
L*	H*	L*	H*	x	x	x	
L*	H*	L*	L*	x	x	x	x—not contiguous bytes
L	L	H	H	H	L	L	
L*	L*	H*	L*	x	x	x	x—not contiguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

BLE # asserted when D0–D7 of 16-bit bus is active.
 BHE # asserted when D8–D15 of 16-bit bus is active.
 A1 low for all even words; A1 high for all odd words.

Key:

- x = don't care
- H = high voltage level
- L = low voltage level
- * = a non-occurring pattern of Byte Enables; either none are asserted, or the pattern has Byte Enables asserted for non-contiguous bytes

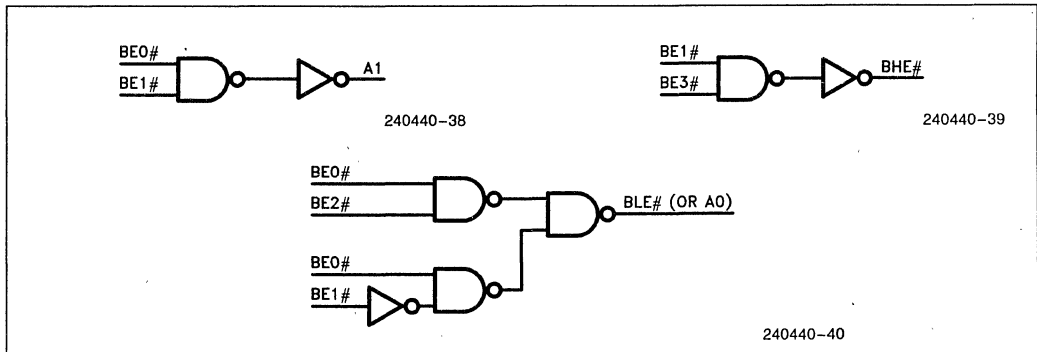
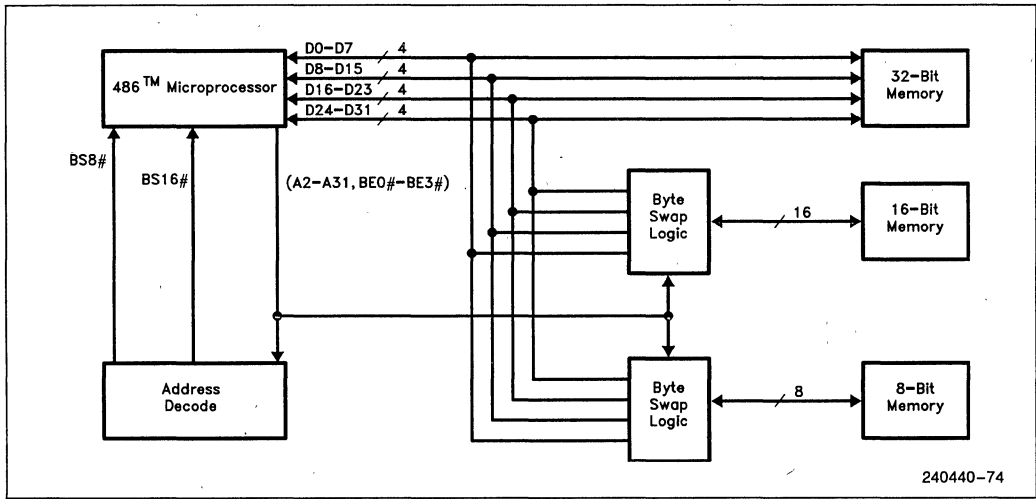


Figure 7.5. Logic to Generate A1, BHE # and BLE # for 16-Bit Busses

Combinations of BE0 #–BE3 # which never occur are those in which two or three asserted byte enables are separated by one or more negated byte enables. These combinations are “don't care” conditions in the decoder. A decoder can use the non-occurring BE0 #–BE3 # combinations to its best advantage.

Figure 7.6 shows a 486 microprocessor data bus interface to 16- and 8-bit wide memories. External byte swapping logic is needed on the data lines so that data is supplied to, and received from the 486 microprocessor on the correct data pins (see Table 7.4).


Figure 7.6. Data Bus Interface to 16- and 8-bit Memories
7.1.5 DYNAMIC BUS SIZING DURING CACHE LINE FILLS

BS8# and BS16# can be driven during cache line fills. The 486 microprocessor will generate enough 8- or 16-bit cycles to fill the cache line. This can be up to 16 8-bit cycles.

The external system should assume that all byte enables are active for the first cycle of a cache line fill. The 486 microprocessor will generate proper byte enables for subsequent cycles in the line fill. Table 7.6 shows the appropriate A0 (BLE#), A1 and BHE# for the various combinations of the 486 microprocessor byte enables on both the first and subsequent cycles of the cache line fill. The "*" marks all combinations of byte enables that will be generated by the 486 microprocessor during a cache line fill.

7.1.6 OPERAND ALIGNMENT

Physical 4-byte words begin at addresses that are multiples of four. It is possible to transfer a logical word of memory or I/O at the expense of extra cycles. Examples are 4-byte operands beginning at addresses that are not evenly divisible by 4, or 2-byte words split between two physical 4-byte words. These are referred to as unaligned transfers.

Operand alignment and data bus size dictate when multiple bus cycles are required. Table 7.7 describes the transfer cycles generated for all combinations of logical operand lengths, alignment, and data bus sizing. When multiple cycles are required to transfer a multi-byte logical operand, the highest-order bytes are transferred first. For example, when the processor does a 4-byte unaligned read beginning at location x11 in the 4-byte aligned space, the three high order bytes are read in the first bus cycle. The low byte is read in a subsequent bus cycle.

Table 7.6. Generating A0, A1 and BHE# from the i486™ Microprocessor Byte Enables

BE3#	BE2#	BE1#	BE0#	First Cache Fill Cycle			Any Other Cycle		
				A0	A1	BHE#	A0	A1	BHE#
1	1	1	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
*0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	1	0	0
1	0	0	1	0	0	0	1	0	0
*0	0	0	1	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	1
*0	0	1	1	0	0	0	0	1	0
*0	1	1	1	0	0	0	1	1	0

second to writes. For example, if a wait state needs to be added to a write, the cycle would be called 2-3.

Basic two clock read and write cycles are shown in Figure 7.7. The 486 microprocessor initiates a cycle by asserting the address status signal (ADS#) at the rising edge of the first clock. The ADS# output indicates that a valid bus cycle definition and address is available on the cycle definition lines and address bus.

The non-burst ready input (RDY#) is returned by the external system in the second clock. RDY# indicates that the external system has presented valid data on the data pins in response to a read or the external system has accepted data in response to a write.

The 486 microprocessor samples RDY# at the end of the second clock. The cycle is complete if RDY# is active (LOW) when sampled. Note that RDY# is ignored at the end of the first clock of the bus cycle.

The burst last signal (BLAST#) is asserted (LOW) by the 486 microprocessor during the second clock of the first cycle in all bus transfers illustrated in Figure 7.7. This indicates that each transfer is complete after a single cycle. The 486 microprocessor asserts BLAST# in the last cycle of a bus transfer.

The timing of the parity check output (PCHK#) is shown in Figure 7.7. The 486 microprocessor drives the PCHK# output one clock after ready terminates a read cycle. PCHK# indicates the parity status for the data sampled at the end of the previous clock. The PCHK# signal can be used by the external system. The 486 microprocessor does nothing in response to the PCHK# output.

7.2.1.2 Inserting Wait States

The external system can insert wait states into the basic 2-2 cycle by driving RDY# inactive at the end of the second clock. RDY# must be driven inactive to insert a wait state. Figure 7.8 illustrates a simple non-burst, non-cacheable signal with one wait state added. Any number of wait states can be added to a 486 microprocessor bus cycle by maintaining RDY# inactive.

The burst ready input (BRDY#) must be driven inactive on all clock edges where RDY# is driven inactive for proper operation of these simple non-burst cycles.

7.2.2 MULTIPLE AND BURST CYCLE BUS TRANSFERS

Multiple cycle bus transfers can be caused by internal requests from the 486 microprocessor or by the external memory system. An internal request for a 64-bit floating point load or a 128-bit pre-fetch must take more than one cycle. Internal requests for unaligned data may also require multiple bus cycles. A cache line fill requires multiple cycles to complete. The external system can cause a multiple cycle transfer when it can only supply 8 or 16 bits per cycle.

Only multiple cycle transfers caused by internal requests are considered in this section. Cacheable cycles and 8- and 16-bit transfers are covered in Sections 7.2.3 and 7.2.5.

7.2.2.1 Burst Cycles

The 486 microprocessor can accept burst cycles for any bus requests that require more than a single data cycle. During burst cycles, a new data item is strobed into the 486 microprocessor every clock rather than every other clock as in non-burst cycles. The fastest burst cycle requires 2 clocks for the first data item with subsequent data items returned every clock.

The 486 microprocessor is capable of bursting a maximum of 32 bits during a write. Burst writes can only occur if BS8# or BS16# is asserted. For example, the 486 microprocessor can burst write four 8-bit operands or two 16-bit operands in a single burst cycle. But the 486 microprocessor cannot burst multiple 32-bit writes in a single burst cycle.

Burst cycles begin with the 486 microprocessor driving out an address and asserting ADS# in the same manner as non-burst cycles. The 486 microprocessor indicates that it is willing to perform a burst cycle by holding the burst last signal (BLAST#) inactive in the second clock of the cycle. The external system indicates its willingness to do a burst cycle by returning the burst ready signal (BRDY#) active.

The addresses of the data items in a burst cycle will all fall within the same 16-byte aligned area (corresponding to an internal 486 microprocessor cache line). A 16-byte aligned area begins at location XXXXXX0 and ends at location XXXXXXF. During a burst cycle, only BE0-3#, A₂, and A₃ may change. A₄-A₃₁, M/IO#, D/C#, and W/R# will remain stable throughout a burst. Given the first address in a burst, external hardware can easily calculate the address of subsequent transfers in advance. An external memory system can be designed to quickly fill the 486 microprocessor internal cache lines.

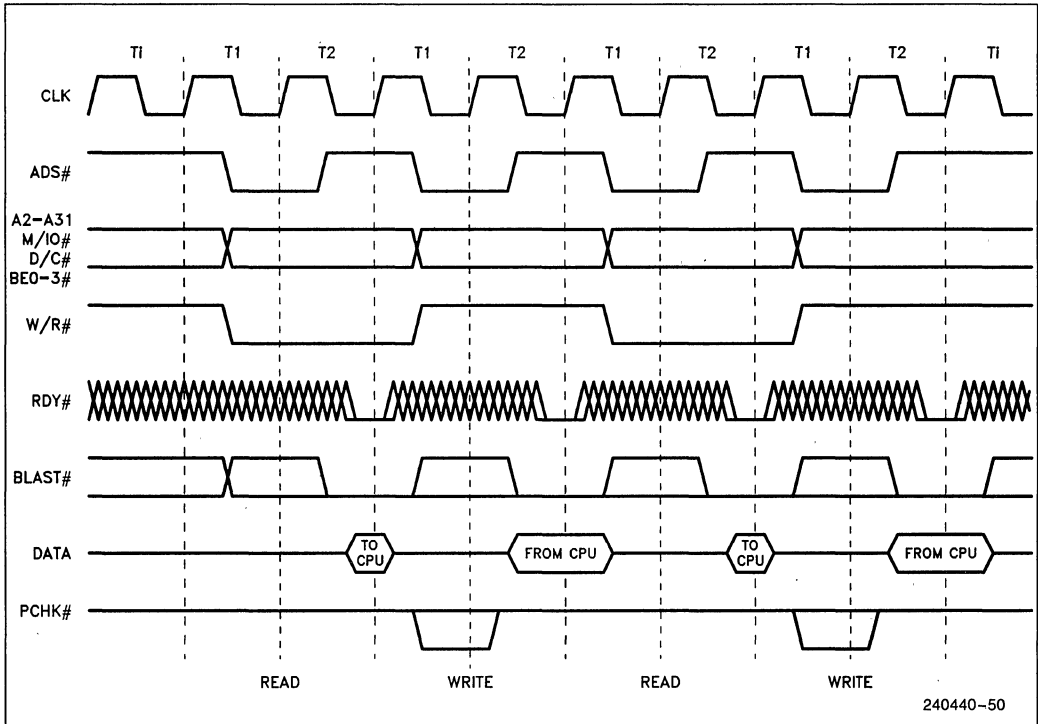


Figure 7.7. Basic 2-2 Bus Cycle

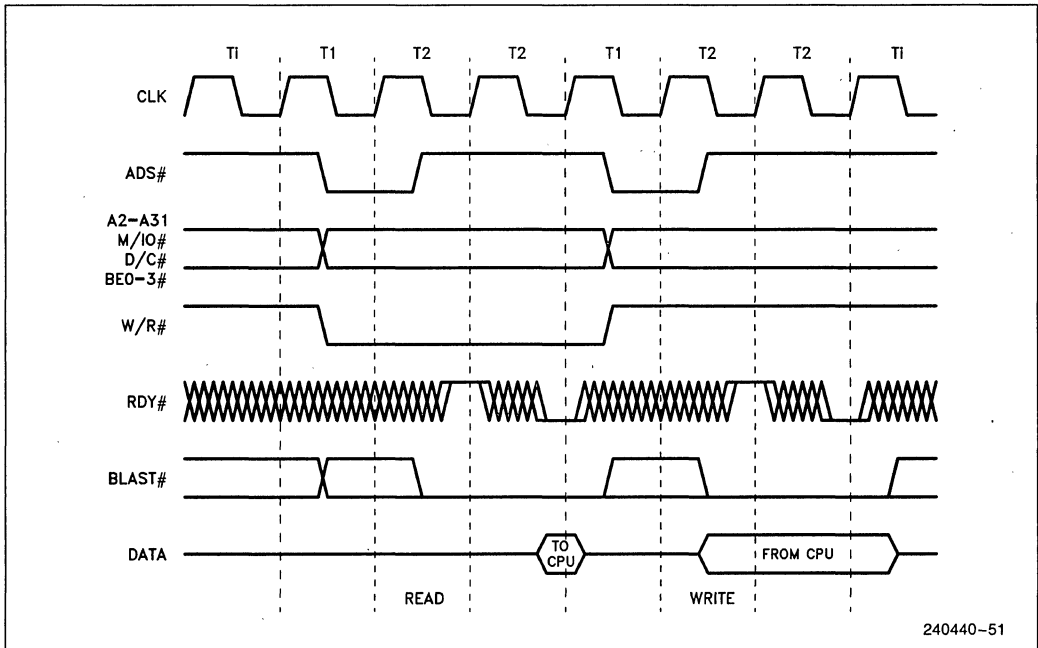


Figure 7.8. Basic 3-3 Bus Cycle

Burst cycles are not limited to cache line fills. Any multiple cycle read request by the 486 microprocessor can be converted into a burst cycle. The 486 microprocessor will only burst the number of bytes needed to complete a transfer. For example, eight bytes will be bursted in for a 64-bit floating point non-cacheable read.

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# (non-burst ready) in the first cycle of a transfer. For cycles that cannot be bursted such as interrupt acknowledge and halt, BRDY# has the same effect as RDY#. BRDY# is ignored if both BRDY# and RDY# are returned in the same clock. Memory areas and peripheral devices that cannot perform bursting must terminate cycles with RDY#.

7.2.2.2 Terminating Multiple and Burst Cycle Transfers

The 486 microprocessor drives BLAST# inactive for all but the last cycle in a multiple cycle transfer. BLAST# is driven inactive in the first cycle to inform the external system that the transfer could take additional cycles. BLAST# is driven active in the last cycle of the transfer indicating that the next time BRDY# or RDY# is returned the transfer is complete.

BLAST# is not valid in the first clock of a bus cycle. It should be sampled only in the second and subsequent clocks when RDY# or BRDY# is returned.

The number of cycles in a transfer is a function of several factors including the number of bytes the microprocessor needs to complete an internal request (1, 2, 4, 8, or 16), the state of the bus size inputs (BS8# and BS16#), the state of the cache enable input (KEN#) and alignment of the data to be transferred.

When the 486 microprocessor initiates a request it knows how many bytes will be transferred and if the data is aligned. The external system must tell the microprocessor whether the data is cacheable (if the transfer is a read) and the width of the bus by returning the state of the KEN#, BS8# and BS16# inputs one clock before RDY# or BRDY# is returned. The 486 microprocessor determines how many cycles a transfer will take based on its internal information and inputs from the external system.

BLAST# is not valid in the first clock of a bus cycle because the 486 microprocessor cannot determine the number of cycles a transfer will take until the external system returns KEN#, BS8# and BS16#. BLAST# should only be sampled in the second and subsequent clocks of a cycle when the external system returns RDY# or BRDY#.

7.2.2.3 Non-Cacheable, Non-Burst, Multiple Cycle Transfers

Figure 7.9 illustrates a 2 cycle non-burst, non-cacheable multiple cycle read. This transfer is simply a sequence of two single cycle transfers. The 486 microprocessor indicates to the external system that this is a multiple cycle transfer by driving BLAST# inactive during the second clock of the first cycle. The external system returns RDY# active indicating that it will not burst the data. The external system also indicates that the data is not cacheable by returning KEN# inactive one clock before it returns RDY# active. When the 486 microprocessor samples RDY# active it ignores BRDY#.

Each cycle in the transfer begins when ADS# is driven active and the cycle is complete when the external system returns RDY# active.

The 486 microprocessor indicates the last cycle of the transfer by driving BLAST# active. The next RDY# returned by the external system terminates the transfer.

7.2.2.4 Non-Cacheable Burst Cycles

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# in the first cycle of the transfer. This is illustrated in Figure 7.10.

There are several features to note in the burst read. ADS# is only driven active during the first cycle of the transfer. RDY# must be driven inactive when BRDY# is returned active.

BLAST# behaves exactly as it does in the non-burst read. BLAST# is driven inactive in the second clock of the first cycle of the transfer indicating more cycles to follow. In the last cycle, BLAST# is driven active telling the external memory system to end the burst after returning the next BRDY#.

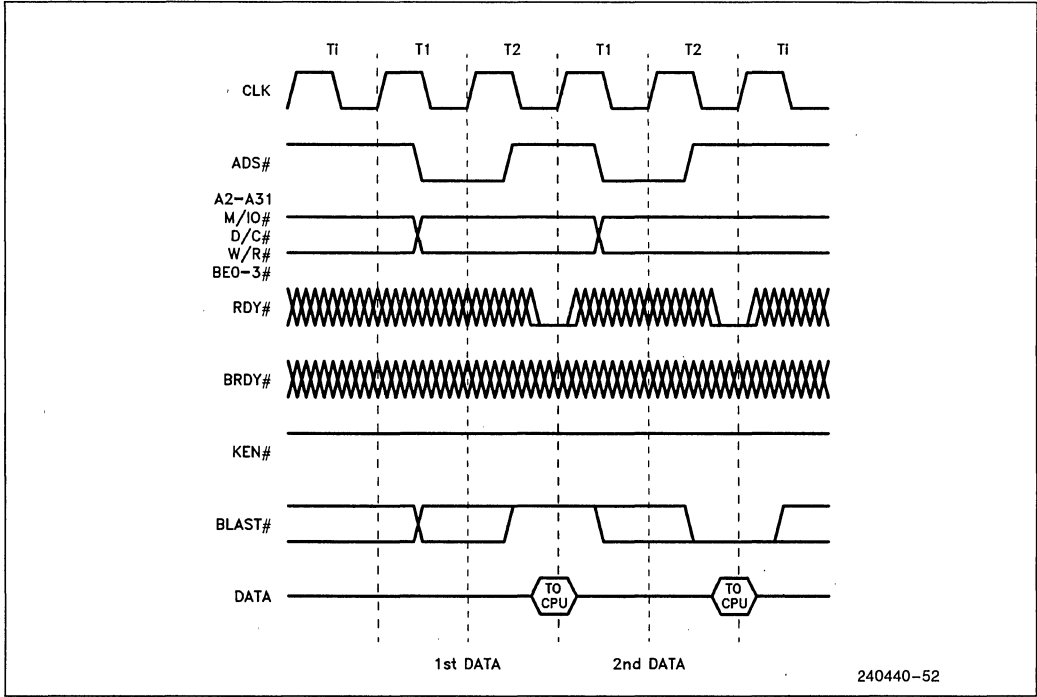


Figure 7.9. Non-Cacheable, Non-Burst, Multiple Cycle Transfers

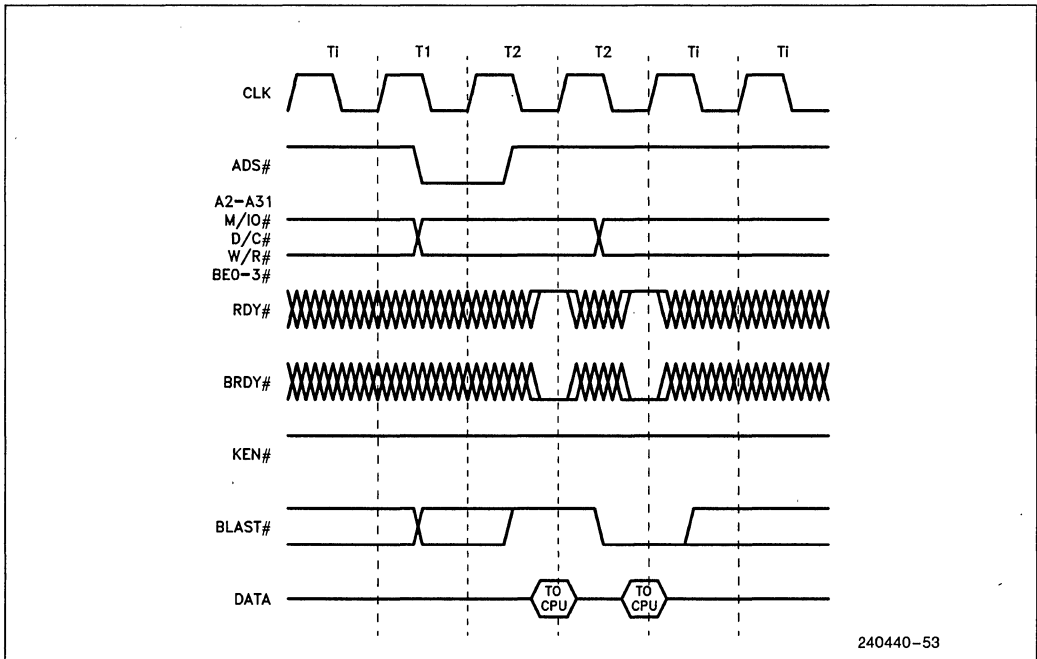


Figure 7.10. Non-Cacheable Burst Cycle

7.2.3 CACHEABLE CYCLES

Any memory read can become a cache fill operation. The external memory system can allow a read request to fill a cache line by returning KEN# active one clock before RDY# or BRDY# during the first cycle of the transfer on the external bus. Once KEN# is asserted and the remaining three requirements described below are met, the 486 microprocessor will fetch an entire cache line regardless of the state of KEN#. KEN# must be returned active in the last cycle of the transfer for the data to be written into the internal cache. The 486 microprocessor will only convert memory reads or prefetches into a cache fill.

KEN# is ignored during write or I/O cycles. Memory writes will only be stored in the on-chip cache if there is a cache hit. I/O space is never cached in the internal cache.

To transform a read or a prefetch into a cache line fill the following conditions must be met:

1. The KEN# pin must be asserted one clock prior to RDY# or BRDY# being returned for the first data cycle.
2. The cycle must be of the type that can be internally cached. (Locked reads, I/O reads, and interrupt acknowledge cycles are never cached).
3. The page table entry must have the page cache disable bit (PCD) set to 0. To cache a page table entry, the page directory must have PCD=0. To cache reads or prefetches when paging is disabled, or to cache the page directory entry, control register 3 (CR3) must have PCD=0.
4. The cache disable (CD) bit in control register 0 (CR0) must be clear.

External hardware can determine when the 486 microprocessor has transformed a read or prefetch into a cache fill by examining the KEN#, M/IO#, D/C#, W/R#, LOCK#, and PCD pins. These pins convey to the system the outcome of conditions 1–3 in the above list. In addition, the 486 drives PCD high whenever the CD bit in CR0 is set, so that external hardware can evaluate condition 4.

Cacheable cycles can be burst or non-burst.

7.2.3.1 Byte Enables during a Cache Line Fill

For the first cycle in the line fill, the state of the byte enables should be ignored. In a non-cacheable memory read, the byte enables indicate the bytes actually required by the memory or code fetch.

The 486 microprocessor expects to receive valid data on its entire bus (32 bits) in the first cycle of a cache line fill. Data should be returned with the assumption that all the byte enable pins are driven active. However if BS8# is asserted only one byte need be returned on data lines D0–D7. Similarly if BS16# is asserted two bytes should be returned on D0–D15.

The 486 microprocessor will generate the addresses and byte enables for all subsequent cycles in the line fill. The order in which data is read during a line fill depends on the address of the first item read. Byte ordering is discussed in Section 7.2.4.

7.2.3.2 Non-Burst Cacheable Cycles

Figure 7.11 shows a non-burst cacheable cycle. The cycle becomes a cache fill when the 486 microprocessor samples KEN# active at the end of the first clock. The 486 microprocessor drives BLAST# inactive in the second clock in response to KEN#. BLAST# is driven inactive because a cache fill requires 3 additional cycles to complete. BLAST# remains inactive until the last transfer in the cache line fill.

Note that this cycle would be a single bus cycle if KEN# was not sampled active at the end of the first clock. The subsequent three reads would not have happened since a cache fill was not requested.

The BLAST# output is invalid in the first clock of a cycle. BLAST# may be active during the first clock due to earlier inputs. Ignore BLAST# until the second clock.

During the first cycle of the cache line fill the external system should treat the byte enables as if they are all active. In subsequent cycles in the burst, the 486 microprocessor drives the address lines and byte enables (see Section 7.2.4.2 for **Burst and Cache Line Fill Order**).

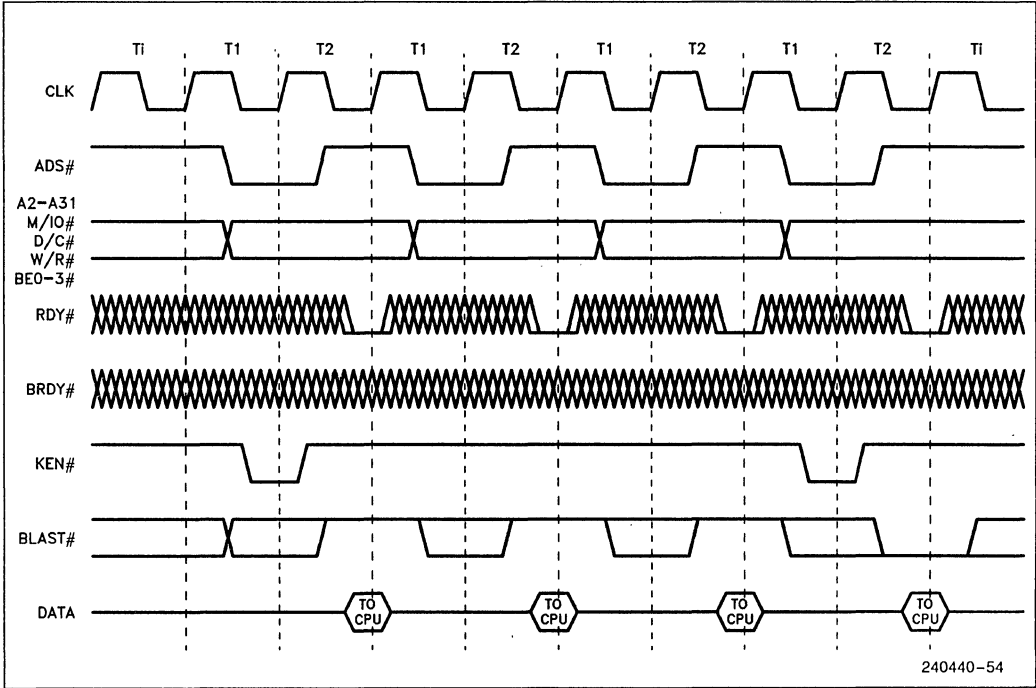


Figure 7.11. Non-Burst, Cacheable Cycles

7.2.3.3 Burst Cacheable Cycles

Figure 7.12 illustrates a burst mode cache fill. As in Figure 7.11, the transfer becomes a cache line fill when the external system returns KEN# active at the end of the first clock in the cycle.

The external system informs the 486 microprocessor that it will burst the line in by driving BRDY# active at the end of the first cycle in the transfer.

Note that during a burst cycle ADS# is only driven with the first address.

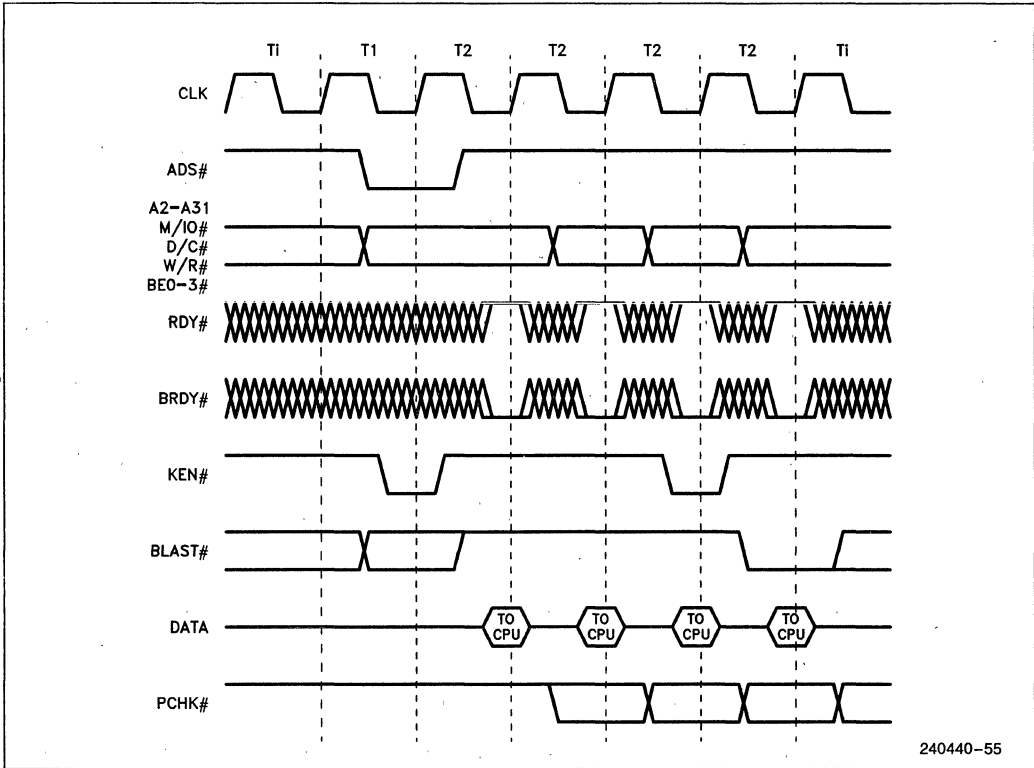


Figure 7.12. Burst Cacheable Cycle

7.2.3.4 Effect of Changing KEN# during a Cache Line Fill

KEN# can change multiple times as long as it arrives at its final value in the clock before RDY# or BRDY# is returned. This is illustrated in Figure 7.13. Note that the timing of BLAST# follows that of KEN# by one clock. In the first clock KEN# is driven active converting the cycle into a cache fill and in the second clock BLAST# is driven inactive in response. In the second clock, KEN# is driven inac-

tive converting the cache fill back to a normal cycle and BLAST# responds by going active in the next clock. Finally in the third clock KEN# goes active again converting the cycle to a cache fill and BLAST# go inactive in the next clock. RDY# is returned active in the fourth clock starting the cache fill.

KEN# can also change multiple times before a burst cycle as long as it arrives at its final value one clock before ready is returned active.

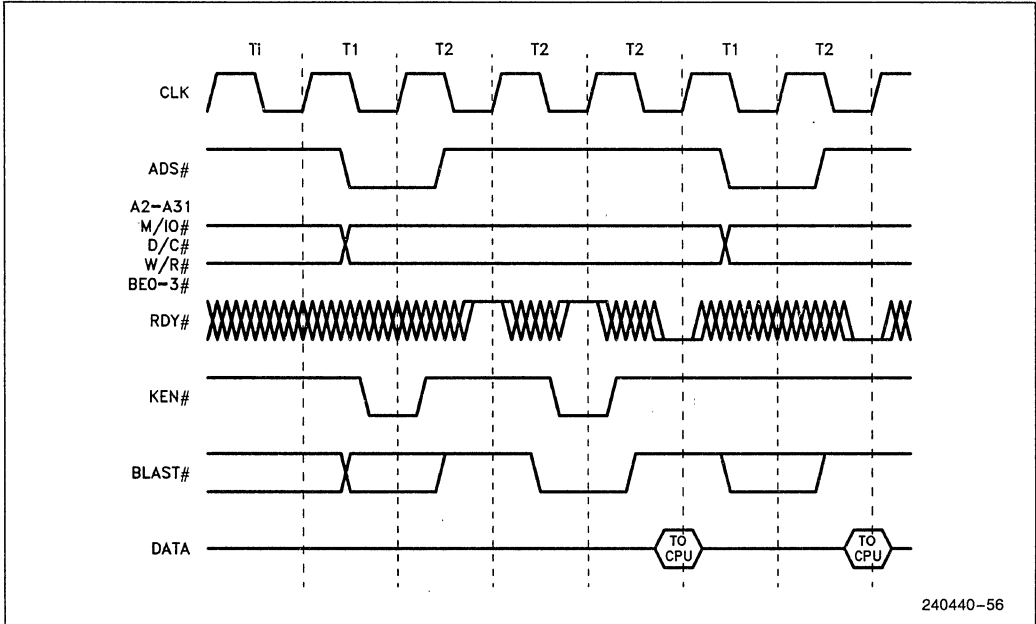


Figure 7.13. Effect of Changing KEN #

7.2.4 BURST MODE DETAILS

7.2.4.1 Adding Wait States to Burst Cycles

Burst cycles need not return data on every clock. The 486 microprocessor will only strobe data into the chip when either RDY# or BRDY# are active.

Driving BRDY# and RDY# inactive adds a wait state to the transfer. A burst cycle where two clocks are required for every burst item is shown in Figure 7.14.

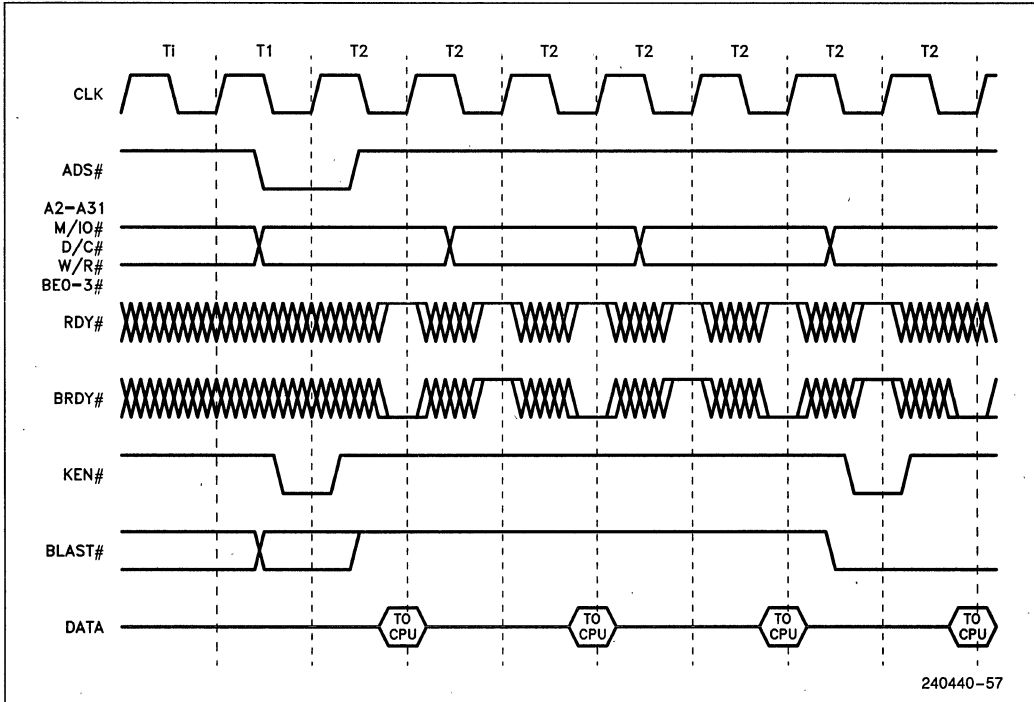


Figure 7.14. Slow Burst Cycle

7.2.4.2 Burst and Cache Line Fill Order

The burst order used by the 486 microprocessor is shown in Table 7.7. Non-burst cache line fills also follow the order in Table 7.7.

The microprocessor presents each request for data in an order determined by the first address in the transfer. For example, if the first address was 104 the next three addresses in the burst will be 100, 10C and 108.

Table 7.7. Burst Order

First Addr.	Second Addr.	Third Addr.	Fourth Addr.
0	4	8	C
4	0	C	8
8	C	0	4
C	8	4	0

An example of burst address sequencing is shown in Figure 7.15.

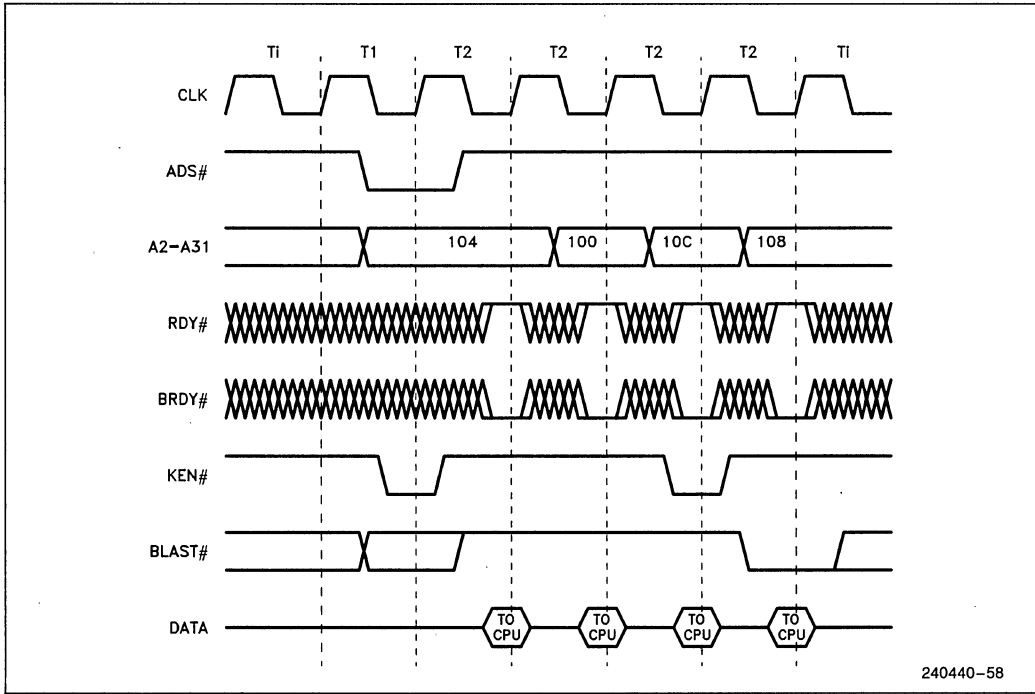


Figure 7.15. Burst Cycle Showing Order of Addresses

The sequences shown in Table 7.7 accommodate systems with 64-bit busses as well as systems with 32-bit data busses. The sequence applies to all bursts, regardless of whether the purpose of the burst is to fill a cache line, do a 64-bit read, or do a pre-fetch. If either BS8# or BS16# is returned active, the 486 microprocessor completes the transfer of the current 32-bit word before progressing to the next 32-bit word. For example, a BS16# burst to address 4 has the following order: 4-6-0-2-C-E-8-A.

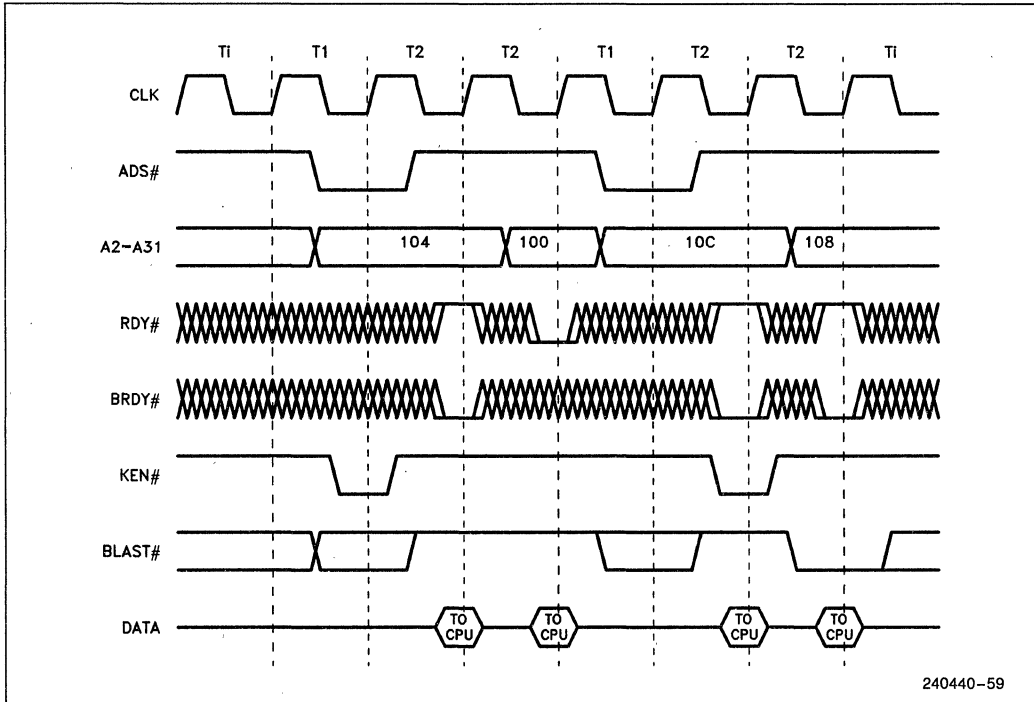
7.2.4.3 Interrupted Burst Cycles

Some memory systems may not be able to respond with burst cycles in the order defined in Table 7.7. To support these systems the 486 microprocessor allows a burst cycle to be interrupted at any time.

The 486 microprocessor will automatically generate another normal bus cycle after being interrupted to complete the data transfer. This is called an interrupted burst cycle. The external system can respond to an interrupted burst cycle with another burst cycle.

The external system can interrupt a burst cycle by returning RDY# instead of BRDY#. RDY# can be returned after any number of data cycles terminated with BRDY#.

An example of an interrupted burst cycle is shown in Figure 7.16. The 486 microprocessor immediately drives ADS# active to initiate a new bus cycle after RDY# is returned active. BLAST# is driven inactive one clock after ADS# begins the second bus cycle indicating that the transfer is not complete.



240440-59

Figure 7.16. Interrupted Burst Cycle

KEN# need not be returned active in the first data cycle of the second part of the transfer in Figure 7.16. The cycle had been converted to a cache fill in the first part of the transfer and the 486 microprocessor expects the cache fill to be completed. Note that the first half and second half of the transfer in Figure 7.16 are each two cycle burst transfers.

The order in which the 486 microprocessor requests operands during an interrupted burst transfer is determined by Table 7.7. Mixing RDY# and BRDY# does not change the order in which operand addresses are requested by the 486 microprocessor.

An example of the order in which the 486 microprocessor requests operands during a cycle in which the external system mixes RDY# and BRDY# is shown in Figure 7.17. The 486 microprocessor initially requests a transfer beginning at location 104. The transfer becomes a cache line fill when the external system returns KEN# active. The first cycle of the cache fill transfers the contents of location 104 and is terminated with RDY#. The 486 microprocessor drives out a new request (by asserting ADS#) to address 100. If the external system terminates the second cycle with BRDY#, the 486 microprocessor will next request/expect address 10C. The correct order is determined by the first cycle in the transfer, which may not be the first cycle in the burst if the system mixes RDY# with BRDY#.

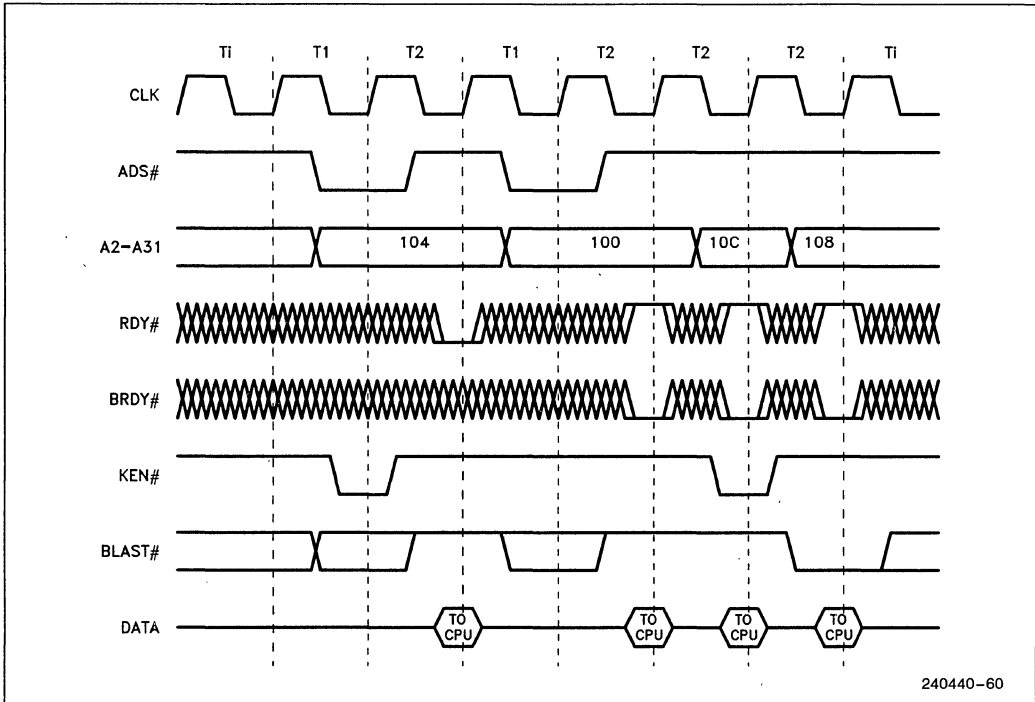


Figure 7.17. Interrupted Burst Cycle with Unobvious Order of Addresses

7.2.5 8- AND 16-BIT CYCLES

The 486 microprocessor supports both 16- and 8-bit external busses through the BS16# and BS8# inputs. BS16# and BS8# allow the external system to specify, on a cycle by cycle basis, whether the addressed component can supply 8, 16 or 32 bits. BS16# and BS8# can be used in burst cycles as well as non-burst cycles. If both BS16# and BS8# are returned active for any bus cycle, the 486 microprocessor will respond as if only BS8# were active.

The timing of BS16# and BS8# is the same as that of KEN#. BS16# and BS8# must be driven active before the first RDY# or BRDY# is driven active.

Driving the BS16# and BS8# active can force the 486 microprocessor to run additional cycles to complete what would have been only a single 32-bit cycle. BS8# and BS16# may change the state of BLAST# when they force subsequent cycles from the transfer.

Figure 7.18 shows an example in which BS8# forces the 486 microprocessor to run two extra cycles to complete a transfer. The 486 microprocessor issues a request for 24 bits of information. The external system drives BS8# active indicating that only eight bits of data can be supplied per cycle. The 486 microprocessor issues two extra cycles to complete the transfer.

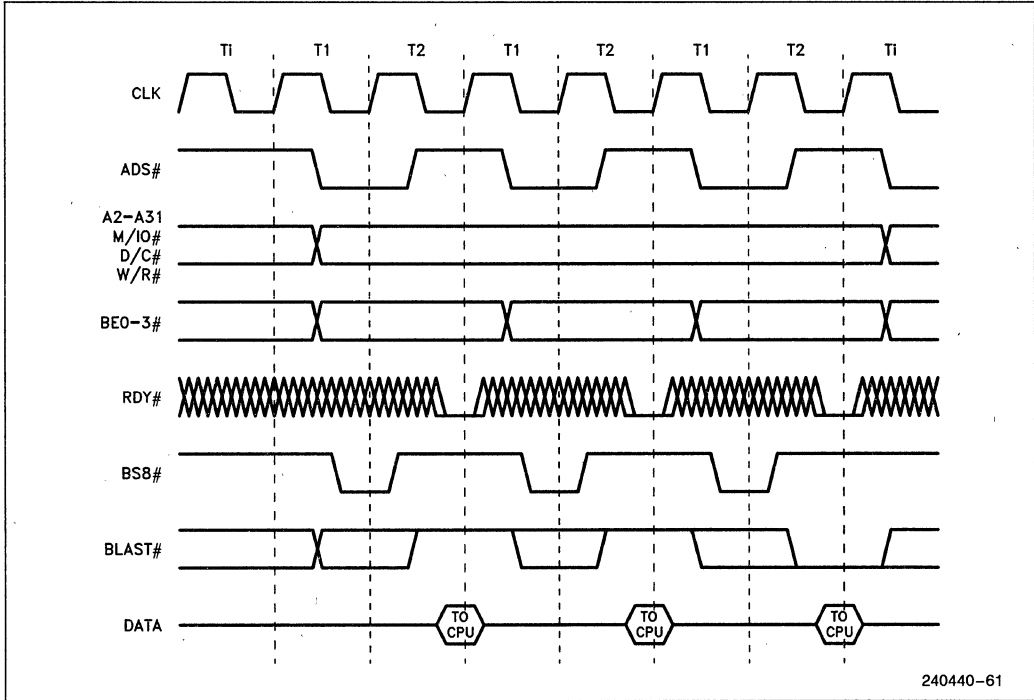


Figure 7.18. 8-Bit Bus Size Cycle

240440-61

Extra cycles forced by the BS16# and BS8# should be viewed as independent bus cycles. BS16# and BS8# should be driven active for each additional cycle unless the addressed device has the ability to change the number of bytes it can return between cycles. The 486 microprocessor will drive BLAST# inactive until the last cycle before the transfer is complete.

BS8# and BS16# operate during burst cycles in exactly the same manner as non-burst cycles. For example, a single non-cacheable read could be transferred by the 486 microprocessor as four 8-bit burst data cycles. Similarly, a single 32-bit write could be written as four 8-bit burst data cycles. An example of a burst write is shown in Figure 7.19. Burst writes can only occur if BS8# or BS16# is asserted.

Refer to Section 7.1.3 for the sequencing of addresses while BS8# or BS16# are active.

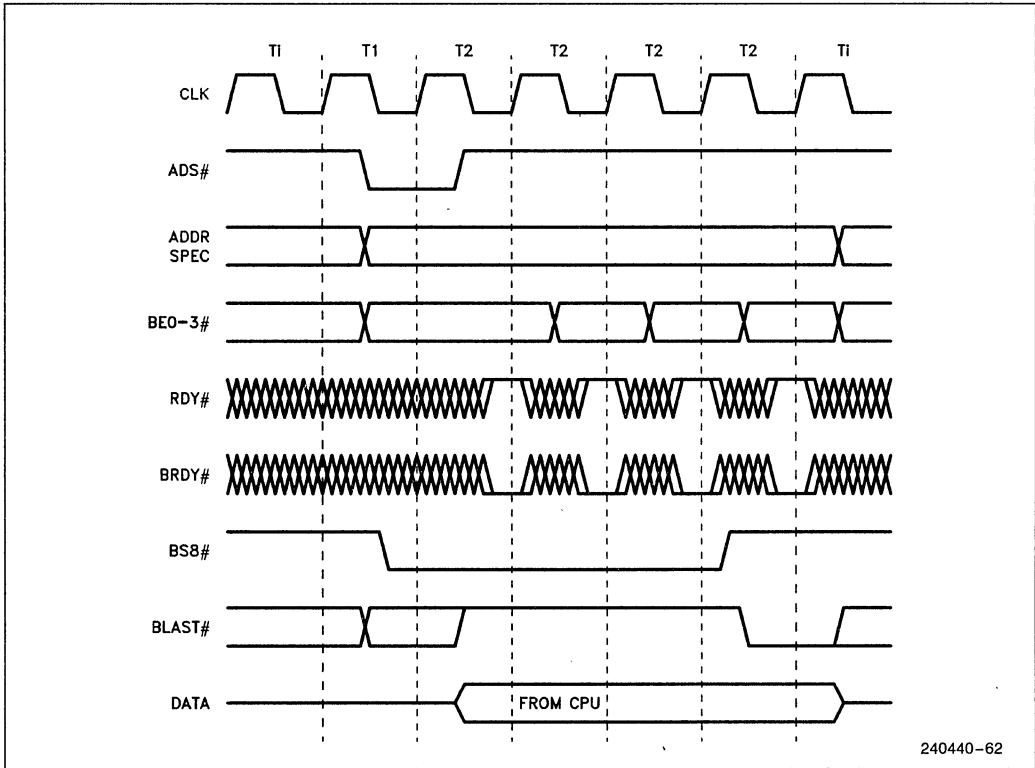


Figure 7.19. Burst Write as a Result of BS8# or BS16#

7.2.6 LOCKED CYCLES

Locked cycles are generated in software for any instruction that performs a read-modify-write operation. During a read-modify-write operation the processor can read and modify a variable in external memory and be assured that the variable is not accessed between the read and write.

Locked cycles are automatically generated during certain bus transfers. The xchg (exchange) instruction generates a locked cycle when one of its operands is memory based. Locked cycles are generated when a segment or page table entry is updated

and during interrupt acknowledge cycles. Locked cycles are also generated when the LOCK instruction prefix is used with selected instructions.

Locked cycles are implemented in hardware with the LOCK# pin. When LOCK# is active, the processor is performing a read-modify-write operation and the external bus should not be relinquished until the cycle is complete. Multiple reads or writes can be locked. A locked cycle is shown in Figure 7.20. LOCK# goes active with the address and bus definition pins at the beginning of the first read cycle and remains active until RDY# is returned for the last write cycle.

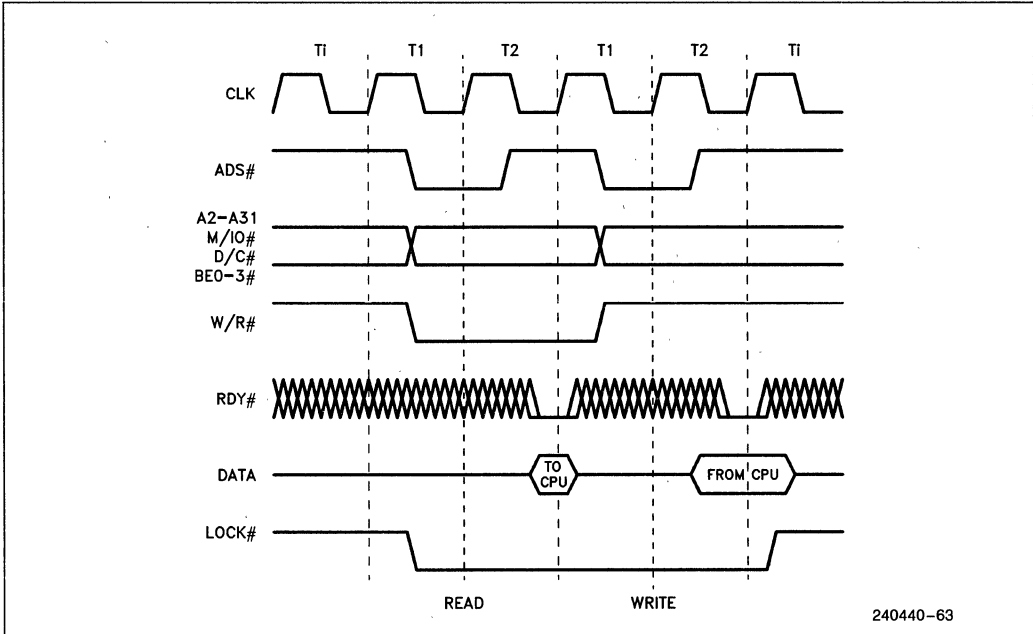


Figure 7.20. Locked Bus Cycle

When LOCK# is active, the 486 microprocessor will recognize address hold and backoff but will not recognize bus hold. It is left to the external system to properly arbitrate a central bus when the 486 microprocessor generates LOCK#.

7.2.7 PSEUDO-LOCKED CYCLES

Pseudo-locked cycles assure that no other master will be given control of the bus during operand transfers which take more than one bus cycle. Examples include 64-bit floating point read and writes, 64-bit descriptor loads and cache line fills.

Pseudo-locked transfers are indicated by the PLOCK# pin. The memory operands must be aligned for correct operation of a pseudo-locked cycle.

PLOCK# need not be examined during burst reads. A 64-bit aligned operand can be retrieved in one burst (note: this is only valid in systems that do not interrupt bursts).

The system must examine PLOCK# during 64-bit writes since the 486 microprocessor cannot burst write more than 32 bits. A 64-bit write will be driven out as two non-burst bus cycles. BLAST# is asserted during both writes since a burst is not possible. PLOCK# is asserted during the first write to indicate that another write follows. This behavior is shown in Figure 7.21.

The first cycle of a 64-bit floating point write is the only case in which both PLOCK# and BLAST# are asserted. Normally PLOCK# and BLAST# are the inverse of each other.

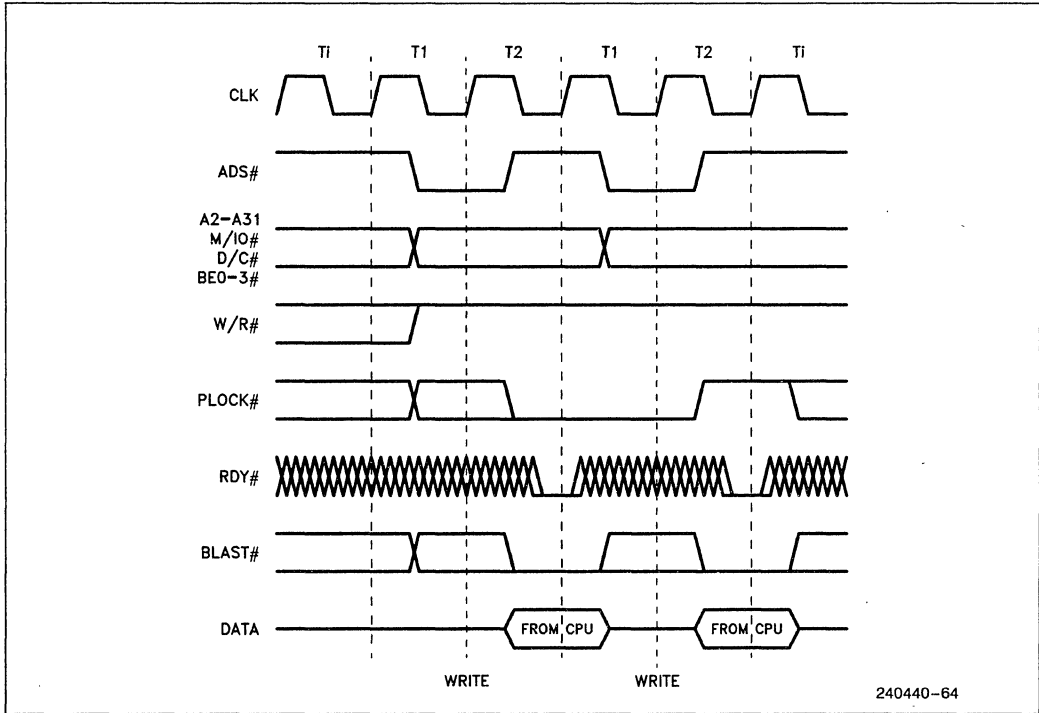


Figure 7.21. Pseudo Lock Timing

PLOCK# can change several times during a cycle settling to its final value in the clock ready is returned.

7.2.8 INVALIDATE CYCLES

Invalidate cycles are needed to keep the 486 microprocessor's internal cache contents consistent with external memory. The 486 microprocessor contains a mechanism for listening to writes by other devices to external memory. When the processor finds a write to a Section of external memory contained in its internal cache, the processor's internal copy is invalidated.

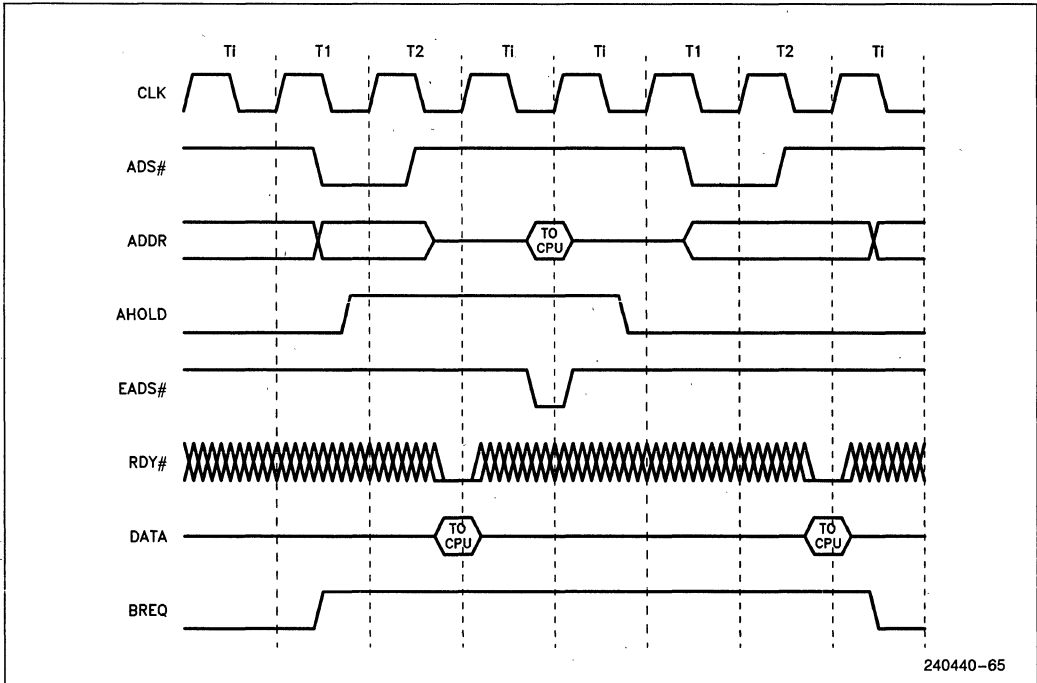
Invalidations use two pins, address hold request (AHOLD) and valid external address (EADS#). There are two steps in an invalidation cycle. First, the external system asserts the AHOLD input forcing the 486 microprocessor to immediately relinquish its address bus. Next, the external system asserts EADS# indicating that a valid address is on the 486 microprocessor's address bus. The microprocessor reads the address over its address lines. If the microprocessor finds this address in its internal cache, the cache entry is invalidated. Note that the 486 microprocessor's address bus is input/output unlike the 386 microprocessor's bus, which is output only.

The 486 microprocessor immediately relinquishes its address bus in the next clock upon assertion of AHOLD. For example, the bus could be 3 wait states into a read cycle. If AHOLD is activated, the 486 microprocessor will immediately float its address bus before ready is returned terminating the bus cycle.

When AHOLD is asserted only the address bus is floated, the data bus can remain active. Data can be returned for a previously specified bus cycle during address hold (see Figures 7.22, 7.23).

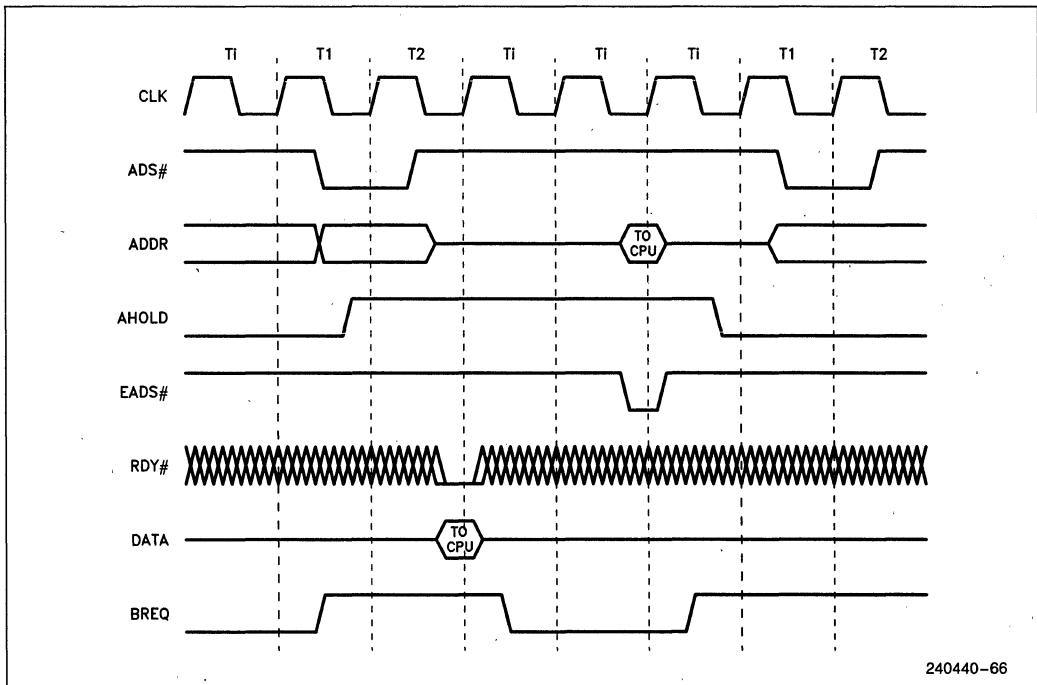
EADS# is normally asserted when an external master drives an address onto the bus. AHOLD need not be driven for EADS# to generate an internal invalidate. If EADS# alone is asserted while the 486 microprocessor is driving the address bus, it is possible that the invalidation address will come from the 486 microprocessor itself.

Running an invalidate cycle prevents the 486 microprocessor cache from satisfying other internal requests, so invalidations should be run only when necessary. The fastest possible invalidate cycle is shown in Figure 7.22, while a more realistic invalidation cycle is shown in 7.23. Both of the examples take one clock of cache access from the rest of the 486 microprocessor.



240440-65

Figure 7.22. Fast Internal Cache Invalidation Cycle



240440-66

Figure 7.23. Typical Internal Cache Invalidation Cycle

7.2.8.1 Rate of Invalidate Cycles

The 486 microprocessor can accept one invalidate per clock except in the last clock of a line fill. One invalidate per clock is possible as long as EADS# is negated in ONE or BOTH of the following cases:

1. In the clock RDY# or BRDY# is returned for the last time.
2. In the clock following RDY# or BRDY# being returned for the last time.

This definition allows two system designs. Simple designs can restrict invalidates to one every other clock. The simple design need not track bus activity. Alternatively, systems can request one invalidate per clock provided that the bus is monitored.

7.2.8.2 Running Invalidate Cycles Concurrently with Line Fills

Precautions are necessary to avoid caching stale data in the 486 microprocessor's cache in a system with a second level cache. An example of a system with a second level cache is shown in Figure 7.24. An external device can be writing to main memory over the system bus while the 486 microprocessor is retrieving data from the second level cache. The 486 microprocessor will need to invalidate a line in its internal cache if the external device is writing to a main memory address also contained in the 486 microprocessor's cache.

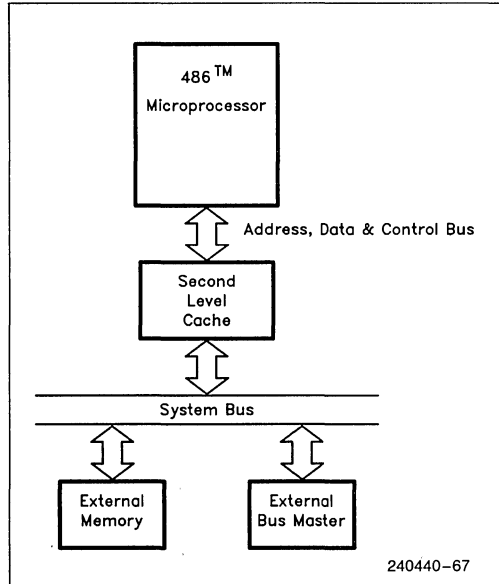


Figure 7.24. System with Second Level Cache

A potential problem exists if the external device is writing to an address in external memory, and at the same time the 486 microprocessor is reading data from the same address in the second level cache. The system must force an invalidation cycle to invalidate the data that the 486 microprocessor has requested during the line fill.

If the system asserts EADS# before the first data in the line fill is returned to the 486 microprocessor, the system must return data consistent with the new data in the external memory upon resumption of the line fill after the invalidation cycle. This is illustrated by the asserted EADS# signal labeled 1 in Figure 7.25.

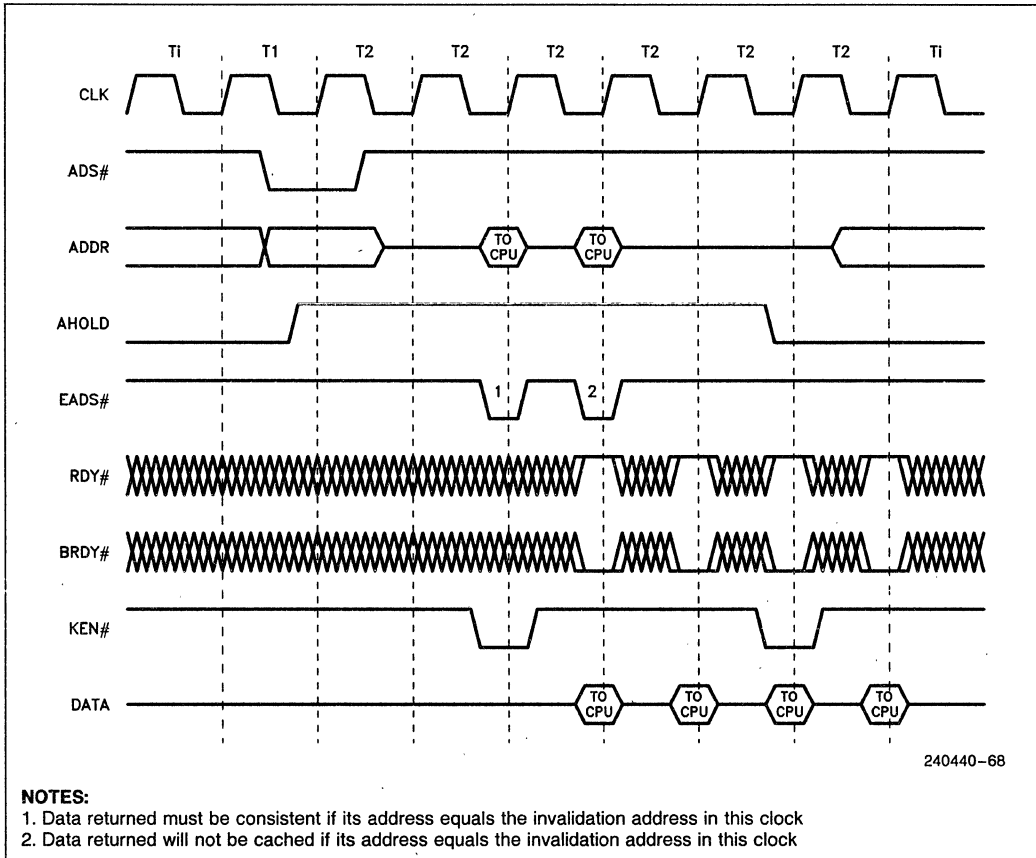


Figure 7.25. Cache Invalidation Cycle Concurrent with Line Fill

If the system asserts EADS# at the same time or after the first data in the line fill is returned (in the same clock that the first RDY# or BRDY# is returned or any subsequent clock in the line fill) the data will be read into the 486 microprocessors input buffers but it will not be stored in the on-chip cache. This is illustrated by asserted EADS# signal labeled 2 in Figure 7.25. The stale data will be used to satisfy the request that initiated the cache fill cycle.

7.2.9 BUS HOLD

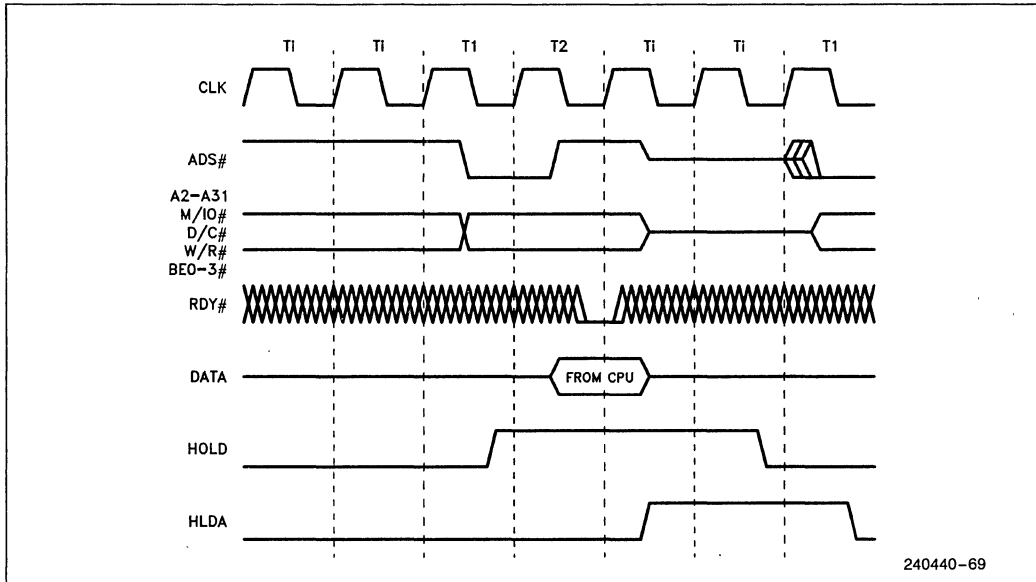
The 486 microprocessor provides a bus hold, hold acknowledge protocol using the bus hold request (HOLD) and bus hold acknowledge (HLDA) pins. Asserting the HOLD input indicates that another bus master desires control of the 486 microprocessor's bus. The processor will respond by floating its bus and driving HLDA active when the current bus cycle,

or sequence of locked cycles is complete. An example of a HOLD/HLDA transaction is shown in Figure 7.26. Unlike the 386 microprocessor, the 486 microprocessor can respond to HOLD by floating its bus and asserting HLDA while RESET is asserted.

The pins floated during bus hold are: BE0# - BE3#, PCD, PWT, W/R#, D/C#, M/IO#, LOCK#, PLOCK#, ADS#, BLAST#, D0-D31, A2-A31, DP0-DP3.

7.2.10 INTERRUPT ACKNOWLEDGE

The 486 microprocessor generates interrupt acknowledge cycles in response to maskable interrupt requests generated on the interrupt request input (INTR) pin. Interrupt acknowledge cycles have a unique cycle type generated on the cycle type pins.

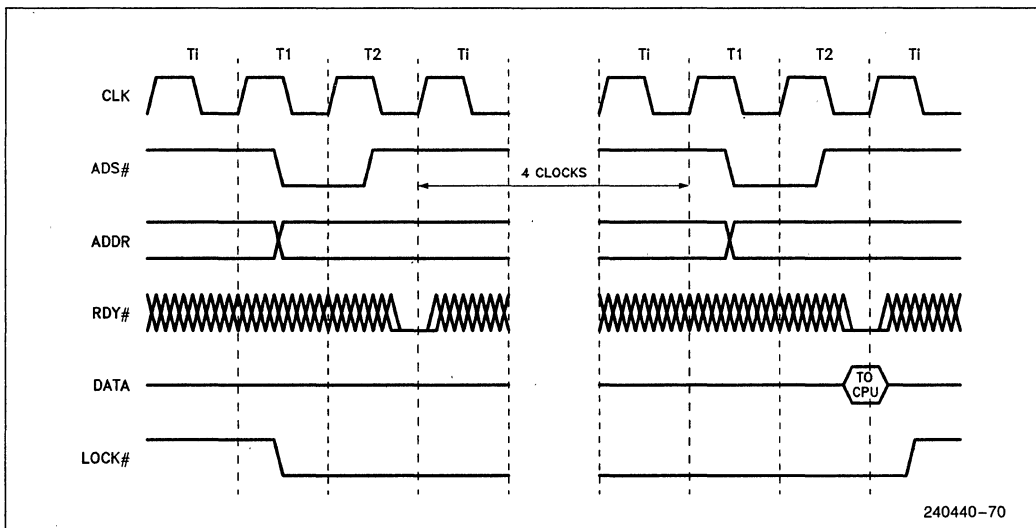


240440-69

Figure 7.26. HOLD/HLDA Cycles

An example interrupt acknowledge transaction is shown in Figure 7.27. Interrupt acknowledge cycles are generated in locked pairs. Data returned during the first cycle is ignored. The interrupt vector is returned during the second cycle on the lower 8 bits of the data bus. The 486 microprocessor has 256 possible interrupt vectors.

The state of A2 distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A31-A3 low, A2 high, BE3#-BE1# high, and BE0# low). The address driven during the second interrupt acknowledge cycle is 0 (A31-A2 low, BE3#-BE1# high, BE0# low).



240440-70

Figure 7.27. Interrupt Acknowledge Cycles

Each of the interrupt acknowledge cycles are terminated when the external system returns RDY# or BRDY#. Wait states can be added by withholding RDY# or BRDY#. The 486 microprocessor automatically generates four idle clocks between the first and second cycles to allow for 8259A recovery time.

7.2.11 SPECIAL BUS CYCLES

The 486 microprocessor provides four special bus cycles to indicate that certain instructions have been executed, or certain conditions have occurred internally. The special bus cycles in Table 7.8 are defined when the bus cycle definition pins are in the following state: M/IO# = 0, D/C# = 0 and W/R# = 1.

Two of the special cycles indicate halt or shutdown. Another special cycle is generated when the 486 microprocessor executes an INVD (invalidate data cache) instruction and could be used to flush an external cache. The Write Back cycle is generated when the 486 microprocessor executes the WBINVD (write-back invalidate data cache) instruction and could be used to synchronize an external write-back cache.

The external hardware must acknowledge these special bus cycles by returning RDY# or BRDY#.

Table 7.8. Special Bus Cycle Encoding

BE3 #	BE2 #	BE1 #	BE0 #	Special Bus Cycle
1	1	1	0	Shutdown
1	1	0	1	Flush
1	0	1	1	Halt
0	1	1	1	Write Back

7.2.12 BUS CYCLE RESTART

In a multi-master system another bus master may require the use of the bus to enable the 486 microprocessor to complete its current bus request. In this situation the 486 microprocessor will need to restart its bus cycle after the other bus master has completed its bus transaction.

A bus cycle may be restarted if the external system asserts the backoff (BOFF#) input. The 486 microprocessor samples the BOFF# pin every clock. The 486 microprocessor will immediately (in the next clock) float its address, data and status pins when BOFF# is asserted (see Figure 7.28). Any bus cycle in progress when BOFF# is asserted is aborted and

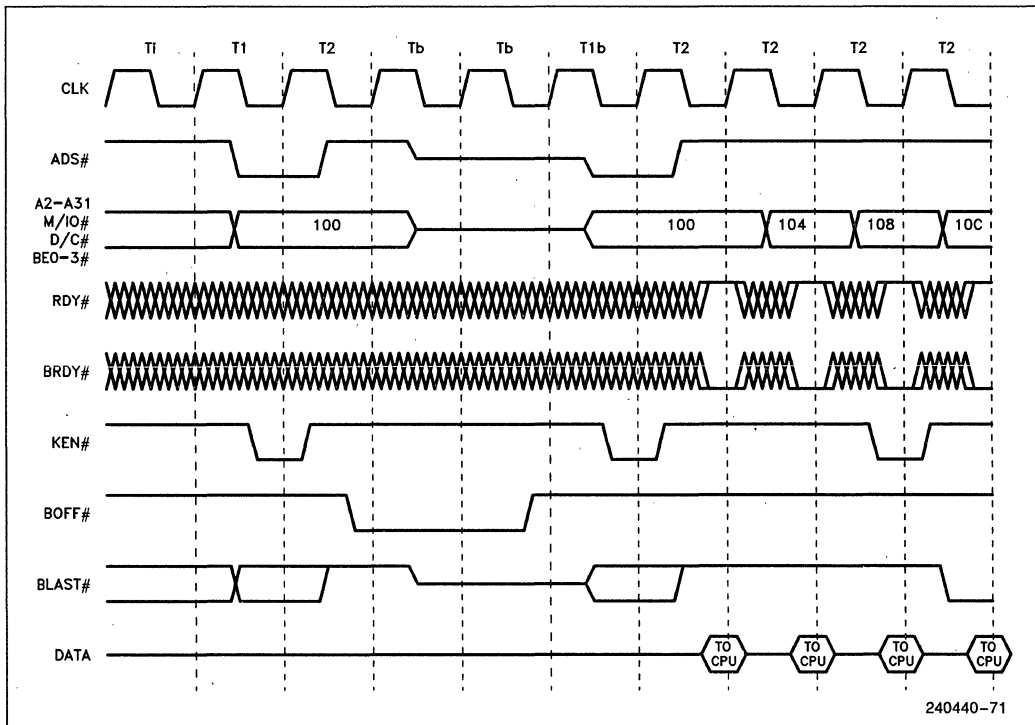


Figure 7.28. Restarted Read Cycle

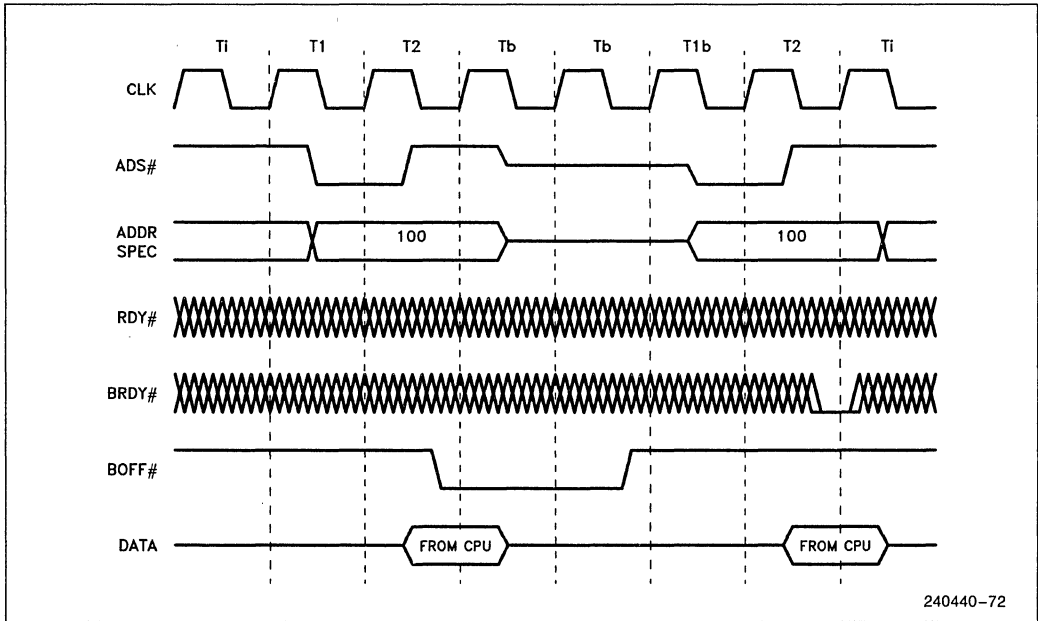


Figure 7.29. Restarted Write Cycle

any data returned to the processor is ignored. The same pins are floated in response to BOFF# as are floated in response to HOLD. HLDA is not generated in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#. If either RDY# or BRDY# are returned in the same clock as BOFF#, BOFF# takes effect.

The device asserting BOFF# is free to run any cycles it wants while the 486 microprocessor bus is in its high impedance state. If backoff is requested after the 486 microprocessor has started a cycle, the new master should wait for memory to return RDY# or BRDY# before assuming control of the bus. Waiting for ready provides a handshake to insure that the memory system is ready to accept a new cycle. If the bus is idle when BOFF# is asserted, the new master can start its cycle two clocks after issuing BOFF#.

The external memory can view BOFF# in the same manner as BLAST#. Asserting BOFF# tells the external memory system that the current cycle is the last cycle in a transfer.

The bus remains in the high impedance state until BOFF# is negated. Upon negation, the 486 microprocessor restarts its bus cycle by driving out the address and status and asserting ADS#. The bus cycle then continues as usual.

Asserting BOFF# during a burst, BS8# or BS16# cycle will force the 486 microprocessor to ignore

data returned for that cycle only. Data from previous cycles will still be valid. For example, if BOFF# is asserted on the third BRDY# of a burst, the 486 microprocessor assumes the data returned with the first and second BRDY#'s is correct and restarts the burst beginning with the third item. The same rule applies to transfers broken into multiple cycle by BS8# or BS16#.

Asserting BOFF# in the same clock as ADS# will cause the 486 microprocessor to float its bus in the next clock and leave ADS# floating low. Since ADS# is floating low, a peripheral may think that a new bus cycle has begun even-though the cycle was aborted. There are two possible solutions to this problem. The first is to have all devices recognize this condition and ignore ADS# until ready comes back. The second approach is to use a "two clock" backoff: in the first clock AHOLD is asserted, and in the second clock BOFF# is asserted. This guarantees that ADS# will not be floating low. This is only necessary in systems where BOFF# may be asserted in the same clock as ADS#.

7.2.13 BUS STATES

A bus state diagram is shown in Figure 7.30. A description of the signals used in the diagram is given in Table 7.9.

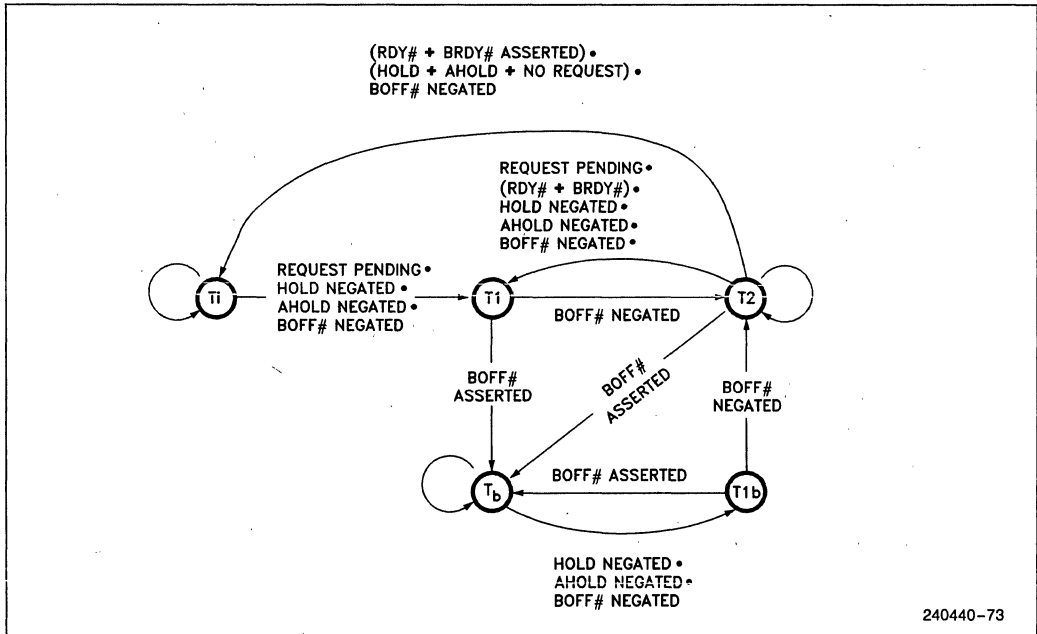


Figure 7.30. Bus State Diagram

Table 7.9. Bus State Description

State	Means
Ti	Bus is idle. Address and status signals may be driven to undefined values, or the bus may be floated to a high impedance state.
T1	First clock cycle of a bus cycle. Valid address and status are driven and ADS# is asserted.
T2	Second and subsequent clock cycles of a bus cycle. Data is driven if the cycle is a write, or data is expected if the cycle is a read. RDY# and BRDY# are sampled.
T1b	First clock cycle of a restarted bus cycle. Valid address and status are driven and ADS# is asserted.
Tb	Second and subsequent clock cycles of an aborted bus cycle.

7.2.14 FLOATING POINT ERROR HANDLING

The 486 microprocessor provides two options for reporting floating point errors. The simplest method is to raise interrupt 16 whenever an unmasked floating point error occurs. This option may be enabled by setting the NE bit in control register 0 (CR0).

The 486 microprocessor also provides the option of allowing external hardware to determine how floating point errors are reported. This option is necessary for compatibility with the error reporting scheme used in DOS based systems. The NE bit must be cleared in CR0 to enable user-defined error reporting. User-defined error reporting is the default condition because the NE bit is cleared on reset.

Two pins, floating point error (FERR#) and ignore numeric error (IGNNE#), are provided to direct the actions of hardware if user-defined error reporting is used. The 486 microprocessor asserts the FERR# output to indicate that a floating point error has occurred. FERR# corresponds to the ERROR# pin on the 387 math coprocessor.

IGNNE# is an input to the 486 microprocessor. When the NE bit in CR0 is cleared, and IGNNE# is asserted, the 486 microprocessor will ignore a user floating point error and continue executing floating point instructions. When IGNNE# is negated, the 486 microprocessor will freeze on floating point instructions which get errors (except for the control instructions FNCLEX, FNINIT, FNSAVE, FNSTENV, FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI and FNSETPM). IGNNE# may be asynchronous to the 486 clock.

In systems with user-defined error reporting, the FERR# pin is connected to the interrupt controller. When an unmasked floating point error occurs, an interrupt is raised. If IGNNE# is high at the time of this interrupt, the 486 microprocessor will freeze (disallowing execution of a subsequent floating point instruction) until the interrupt handler is invoked. By driving the IGNNE# pin low (when clearing the interrupt request), the interrupt handler can allow execution of a floating point instruction, within the interrupt handler, before the error condition is cleared (by FNCLEX, FNINIT, FNSAVE or FNSTENV). If execution of a non-control floating point instruction, within the floating point interrupt handler, is not needed, the IGNNE# pin can be tied HIGH.

8.0 TESTABILITY

Testing in the 486 microprocessor can be divided into two categories: Built-in Self Test (BIST) and external testing. The BIST tests the non-random logic, control ROM (CROM), translation lookaside buffer (TLB) and on-chip cache memory. External tests can be run on the TLB and the on-chip cache. The 486 microprocessor also has a test mode in which all outputs are tristated.

8.1 Built-In Self Test (BIST)

The BIST is initiated by holding the AHOLD (address hold) pin HIGH in the clock prior to RESET going from HIGH to LOW as shown in Figure 6.3. The BIST takes approximately 2**20 clocks, or approximately 42 milliseconds with a 25 MHz 486 microprocessor. No bus cycles will be run by the 486 microprocessor until the BIST is concluded.

The results of BIST is stored in the EAX register. The 486 microprocessor has successfully passed the BIST if the contents of the EAX register are zero. If the results in EAX are not zero then the BIST has detected a flaw in the microprocessor. The microprocessor performs reset and begins normal operation at the completion of the BIST.

The non-random logic, control ROM, on-chip cache and translation lookaside buffer (TLB) are tested during the BIST.

The cache portion of the BIST verifies that the cache is functional and that it is possible to read and write to the cache. The BIST manipulates test registers TR3, TR4 and TR5 while testing the cache. These test registers are described in Section 8.2.

The cache testing algorithm writes a value to each cache entry, reads the value back, and checks that the correct value was read back. The algorithm may be repeated more than once for each of the 512 cache entries using different constants.

The TLB portion of the BIST verifies that the TLB is functional and that it is possible to read and write to the TLB. The BIST manipulates test registers TR6 and TR7 while testing the TLB. TR6 and TR7 are described in Section 8.3.

The 486 microprocessor contains a cache fill buffer and a cache read buffer. For testability writes, data must be written to the cache fill buffer before it can be written to a location in the cache. Data must be read from a cache location into the cache read buffer before the microprocessor can access the data. The cache fill and cache read buffer are both 128 bits wide.

8.2.1 CACHE TESTING REGISTERS TR3, TR4 AND TR5

Figure 8.1 shows the three cache testing registers: the Cache Data Test Register (TR3), the Cache Status Test Register (TR4) and the Cache Control Test Register (TR5). External access to these registers is provided through MOV reg,TREG and MOV TREG, reg instructions.

8.2 On-Chip Cache Testing

The on-chip cache testability hooks are designed to be accessible during the BIST and for assembly language testing of the cache.

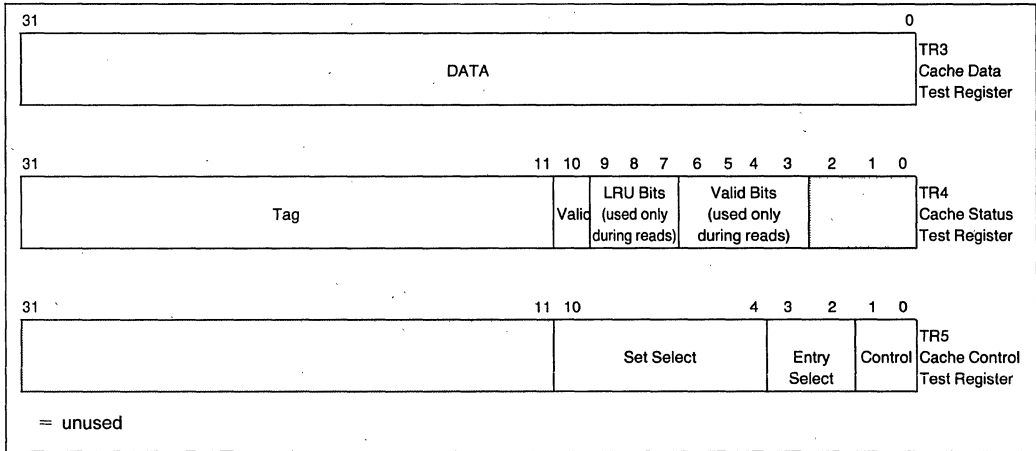


Figure 8.1. Cache Test Registers

Cache Data Test Register: TR3

The cache fill buffer and the cache read buffer can only be accessed through TR3. Data to be written to the cache fill buffer must first be written to TR3. Data read from the cache read buffer must be loaded into TR3.

TR3 is 32 bits wide while the cache fill and read buffers are 128 bits wide. 32 bits of data must be written to TR3 four times to fill the cache fill buffer. 32 bits of data must be read from TR3 four times to empty the cache read buffer. The entry select bits in TR5 determine which 32 bits of data TR3 will access in the buffers.

Cache Status Test Register: TR4

TR4 handles tag, LRU and valid bit information during cache tests. TR4 must be loaded with a tag and a valid bit before a write to the cache. After a read from a cache entry, TR4 contains the tag and valid bit from that entry, and the LRU bits and four valid bits from the accessed set.

Cache Control Test Register: TR5

TR5 specifies which testability operation will be performed and the set and entry within the set which will be accessed.

The seven bit set select field determines which of the 128 sets will be accessed.

The functionality of the two entry select bits depend on the state of the control bits. When the fill or read

buffers are being accessed, the entry select bits point to the 32-bit location in the buffer being accessed. When a cache location is specified, the entry select bits point to one of the four entries in a set. Refer to Table 8.1.

Five testability functions can be performed on the cache. The two control bits in TR5 specify the operation to be executed. The five operations are:

1. Write cache fill buffer
2. Perform a cache testability write
3. Perform a cache testability read
4. Read the cache read buffer
5. Perform a cache flush

Table 8.1 shows the encoding of the two control bits in TR5 for the cache testability functions. Table 8.1 also shows the functionality of the entry and set select bits for each control operation.

The cache tests attempt to use as much of the normal operating circuitry as possible. Therefore when cache tests are being performed, the cache must be disabled (the CD and NW bits in control register must be set to 1 to disable the cache. See Section 5).

8.2.2 CACHE TESTABILITY WRITE

A testability write to the cache is a two step process. First the cache fill buffer must be loaded with 128 bits of data and TR4 loaded with the tag and valid bit. Next the contents of the fill buffer are written to a cache location. Sample assembly code to do a write is given in Figure 8.2.

Table 8.1. Cache Control Bit Encoding and Effect of Control Bits on Entry Select and Set Select Functionality

Control Bits		Operation	Entry Select Bits Function	Set Select Bits
Bit 1	Bit 0			
0	0	Enable { Fill Buffer Write Read Buffer Read	Select 32-bit location in fill/read buffer	—
0	1	Perform Cache Write	Select an entry in set.	Select a set to write to
1	0	Perform Cache Read	Select an entry in set.	Select a set to read from
1	1	Perform Flush Cache	—	—

Sample Assembly Code

An example assembly language sequence to perform a cache write is:

```

;
; eax. ebx. ecx. edx contain the cache line to write
; edi contains the tag information to load
; CRO already says to enable reads/write to TR5
;
; fill the cache buffer
    mov esi,0           ; set up command
    mov tr5,esi        ; load to TR5
    mov tr3,eax        ; load data into cache fill buffer
    mov esi,4
    mov tr5,esi
    mov tr3,ebx
    mov esi,8
    mov tr5,esi
    mov tr3,ecx
    mov esi,0ch
    mov tr5,esi
    mov tr3,edx
;
; load the Cache Status Register
;
    mov tr4,edi        ; load 21-bit tag and valid bit
;
; perform the cache write
;
    mov esi,1
    mov tr5,esi        ; write the cache (set 0, entry 0)

```

An example assembly language sequence to perform a cache read is:

```

;
; data into eax, ebx, ecx, edx; status into edi
;
; read the cache line back
;
    mov esi,2
    mov tr5,esi        ; do cache testability read (set 0, entry 0)
;
; read the data from the read buffer
;
    mov esi,0
    mov tr5,esi
    mov eax,tr3
    mov esi,4
    mov tr5,esi
    mov ebx,tr3
    mov esi,8
    mov tr5,esi
    mov ecx,tr3
    mov esi,0ch
    mov tr5,esi
    mov edx,tr3
;
; read the status from TR4
;
    mov edi,tr4

```

Figure 8.2 Sample Assembly Code for Cache Testing

Loading the fill buffer is accomplished by first writing to the entry select bits in TR5 and setting the control bits in TR5 to 00. The entry select bits identify one of four 32-bit locations in the cache fill buffer to put 32 bits of data. Following the write to TR5, TR3 is written with 32 bits of data which are immediately placed in the cache fill buffer. Writing to TR3 initiates the write to the cache fill buffer. The cache fill buffer is loaded with 128 bits of data by writing to TR5 and TR3 four times using a different entry select location each time.

TR4 must be loaded with the 21-bit tag and valid bit (bit 10 in TR4) before the contents of the fill buffer are written to a cache location.

The contents of the cache fill buffer are written to a cache location by writing TR5 with a control field of 01 along with the set select and entry select fields. The set select and entry select field indicate the location in the cache to be written. The normal cache LRU update circuitry updates the internal LRU bits for the selected set.

Note that a cache testability write can only be done when the cache is disabled for replaces (the CD bit is control register 0 is reset to 1). Also note that care must be taken when directly writing to entries in the cache. If the entry is set to overlap an area of memory that is being used in external memory, that cache entry could inadvertently be used instead of the external memory. Of course, this is exactly the type of operation that one would desire if the cache were to be used as a high speed RAM.

8.2.3 CACHE TESTABILITY READ

A cache testability read is a two step process. First the contents of the cache location are read into the cache read buffer. Next the data is examined by reading it out of the read buffer. Sample assembly code to do a testability read is given in Figure 8.5.

Reading the contents of a cache location into the cache read buffer is initiated by writing TR5 with the control bits set to 10 and the desired seven-bit set select and two-bit entry select. In response to the write to TR5, TR4 is loaded with the 21-bit tag field and the single valid bit from the cache entry read. TR4 is also loaded with the three LRU bits and four valid bits corresponding to the cache set that was accessed. The cache read buffer is filled with the 128-bit value which was found in the data array at the specified location.

The contents of the read buffer are examined by performing four reads of TR3. Before reading TR3 the entry select bits in TR5 must be loaded to indicate which of the four 32-bit words in the read buffer to

transfer into TR3 and the control bits in TR5 must be loaded with 00. The register read of TR3 will initiate the transfer of the 32-bit value from the read buffer to the specified general purpose register.

Note that it is very important that the entire 128-bit quantity from the read buffer and also the information from TR4 be read before any memory references are allowed to occur. If memory operations are allowed to happen, the contents of the read buffer will be corrupted. This is because the testability operations use hardware that is used in normal memory accesses for the 486 microprocessor whether the cache is enabled or not.

8.2.4 FLUSH CACHE

The control bits in TR5 must be written with 11 to flush the cache. None of the other bits in TR5 have any meaning when 11 is written to the control bits. Flushing the cache will reset the LRU bits and the valid bits to 0, but will not change the cache tag or data arrays.

When the cache is flushed by writing to TR5 the special bus cycle indicating a cache flush to the external system is not run (see Section 7.2.11, Special Bus Cycles). The cache should be flushed with the instruction **INVD** (Invalidate Data Cache) instruction or the **WBINVD** (Write-back and Invalidate Data Cache) instruction.

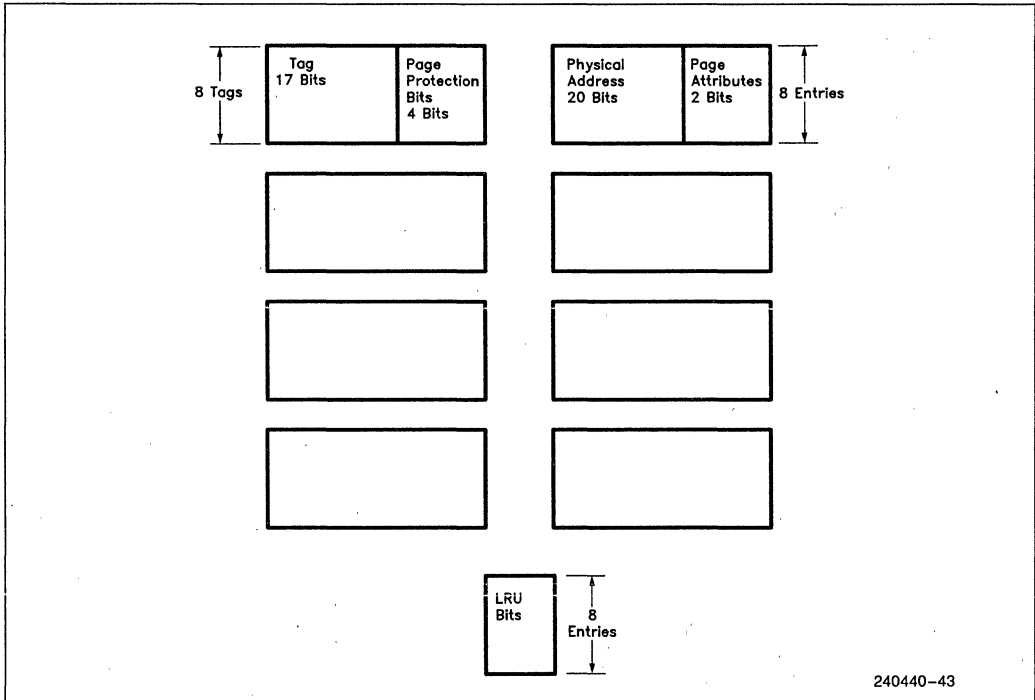
8.3 Translation Lookaside Buffer (TLB) Testing

The 486 microprocessor TLB testability hooks are similar to those in the 386 microprocessor. The testability hooks have been enhanced to provide added test features and to include new features in the 486 microprocessor. The TLB testability hooks are designed to be accessible during the BIST and for assembly language testing of the TLB.

8.3.1 TRANSLATION LOOKASIDE BUFFER ORGANIZATION

The 486 microprocessors TLB is 4-way set associative and has space for 32 entries. The TLB is logically split into three blocks shown in Figure 8.3.

The data block is physically split into four arrays, each with space for eight entries. An entry in the data block is 22 bits wide containing a 20-bit physical address and two bits for the page attributes. The page attributes are the PCD (page cache disable) bit and the PWT (page write-through) bit. Refer to Section 4.5.4 for a discussion of the PCD and PWT bits.



240440-43

Figure 8.3. TLB Organization

The tag block is also split into four arrays, one for each of the data arrays. A tag entry is 21 bits wide containing a 17-bit linear address and four protection bits. The protection bits are valid (V), user/supervisor (U/S), read/write (R/W) and dirty (D).

The third block contains eight three bit quantities used in the pseudo least recently used (LRU) replacement algorithm. These bits are called the LRU bits. The LRU replacement algorithm used in the

TLB is the same as used by the on-chip cache. For a description of this algorithm refer to Section 5.5.

8.3.2 TLB TEST REGISTERS TR6 AND TR7

The two TLB test registers are shown in Figure 8.4. TR6 is the command test register and TR7 is the data test register. External access to these registers is provided through MOV reg,TREG and MOV TREG,reg instructions.

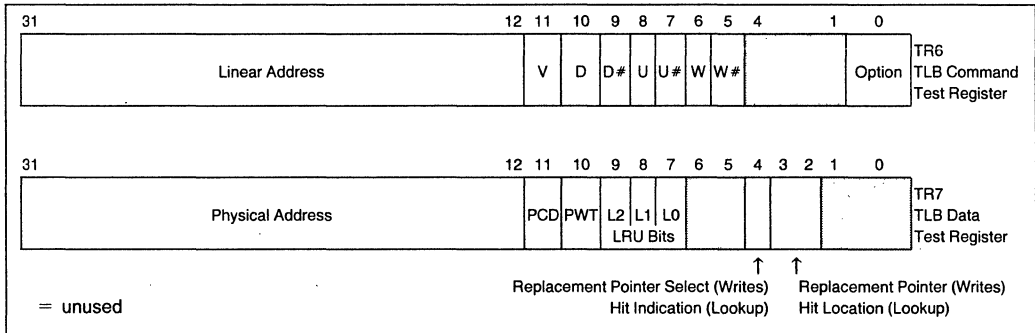


Figure 8.4. TLB Test Registers

Command Test Register: TR6

TR6 contains the tag information and control information used in a TLB test. Loading TR6 with tag and control information initiates a TLB write or lookup test.

TR6 contains three bit fields, a 20-bit linear address (bits 12–31), seven bits for the TLB tag protection bits (bits 5–11) and one bit (bit 0) to define the type of operation to be performed on the TLB.

The 20-bit linear address forms the tag information used in the TLB access. The lower three bits of the linear address select which of the eight sets are accessed. The upper 17 bits of the linear address form the tag stored in the tag array.

The seven TLB tag protection bits are described below.

- V: The valid bit for this TLB entry
- D,D#: The dirty bit for/from the TLB entry
- U,U#: The user/supervisor bit for/from the TLB entry
- W,W#: The read/write bit for/from the TLB entry

Two bits are used to represent the D, U/S and R/W bits in the TLB tag to permit the option of a forced miss or hit during a TLB lookup operation. The forced miss or hit will occur regardless of the state of the actual bit in the TLB. The meaning of these pairs of bits is given in Table 8.2.

The operation bit in TR6 determines if the TLB test operation will be a write or a lookup. The function of the operation bit is given in Table 8.3.

Table 8.3. TR6 Operation Bit Encoding

TR6 Bit 0	TLB Operation to Be Performed
0	TLB Write
1	TLB Lookup

Data Test Register: TR7

TR7 contains the information stored or read from the data block during a TLB test operation. Before a TLB

test write, TR7 contains the physical address and the page attribute bits to be stored in the entry. After a TLB test lookup hit, TR7 contains the physical address, page attributes, LRU bits and entry location from the access.

TR7 contains a 20-bit physical address (bits 12–31), two bits for PCD (bit 11) and PWT (bit 10) and three bits for the LRU bits (bits 7–9). The LRU bits in TR7 are only used during a TLB lookup test. The functionality of TR7 bit 4 differs for TLB writes and lookups. The encoding of bit 4 is defined in Tables 8.4 and 8.5. Finally TR7 contains two bits (bits 2–3) to specify a TLB replacement pointer or the location of a TLB hit.

Table 8.4. Encoding of Bit 4 of TR7 on Writes

TR7 Bit 4	Replacement Pointer Used on TLB Write
0	Pseudo-LRU Replacement Pointer
1	Data Test Register Bits 3:2

Table 8.5. Encoding of Bit 4 of TR7 on Lookups

TR7 Bit 4	Meaning after TLB Lookup Operation
0	TLB Lookup Resulted in a Miss
1	TLB Lookup Resulted in a Hit

A replacement pointer is used during a TLB write. The pointer indicates which of the four entries in an accessed set is to be written. The replacement pointer can be specified to be the internal LRU bits or bits 2–3 in TR7. The source of the replacement pointer is specified by TR7 bit 4. The encoding of bit 4 during a write is given by Table 8.4.

Note that both testability writes and lookups affect the state of the internal LRU bits regardless of the replacement pointer used. All TLB write operations (testability or normal operation) cause the written entry to become the most recently used. For example, during a testability write with the replacement pointer specified by TR7 bits 2–3, the indicated entry is written and that entry becomes the most recently used as specified by the internal LRU bits.

Table 8.2. Meaning of a Pair of TR6 Protection Bits

TR6 Protection Bit (B)	TR6 Protection Bit # (B#)	Meaning on TLB Write Operation	Meaning on TLB Lookup Operation
0	0	Undefined	Miss any TLB TAG Bit B
0	1	Write 0 to TLB TAG Bit B	Match TLB TAG Bit B if 0
1	0	Write 1 to TLB TAG Bit B	Match TLB TAG Bit B if 1
1	1	Undefined	Match any TLB TAG Bit B

There are two TLB testing operations: write entries into the TLB, and perform TLB lookups. One major enhancement over TLB testing in the 386 microprocessor is that paging need not be disabled while executing testability writes or lookups.

Note that any time one TLB set contains the same linear address in more than one of its entries, looking up that linear address will not result in a hit. Therefore a single linear address should not be written to one TLB set more than once.

8.3.3 TLB WRITE TEST

To perform a TLB write TR7 must be loaded followed by a TR6 load. The register operations must be performed in this order since the TLB operation is triggered by the write to TR6.

TR7 is loaded with a 20-bit physical address and values for PCD and PWT to be written to the data portion of the TLB. In addition, bit 4 of TR7 must be loaded to indicate whether to use TR7 bits 3-2 or the internal LRU bits as the replacement pointer on the TLB write operation. Note that the LRU bits in TR7 are not used in a write test.

TR6 must be written to initiate the TLB write operation. Bit 0 in TR6 must be reset to zero to indicate a TLB write. The 20-bit linear address and the seven page protection bits must also be written in TR6 to specify the tag portion of the TLB entry. Note that the three least significant bits of the linear address specify which of the eight sets in the data block will be loaded with the physical address data. Thus only 17 of the linear address bits are stored in the tag array.

8.3.4 TLB LOOKUP TEST

To perform a TLB lookup it is only necessary to write the proper tags and control information into TR6. Bit 0 in TR6 must be set to 1 to indicate a TLB lookup. TR6 must be loaded with a 20-bit linear address and the seven protection bits. To force misses and matches of the individual protection bits on TLB lookups, set the seven protection bits as specified in Table 8.2.

A TLB lookup operation is initiated by the write to TR6. TR7 will indicate the result of the lookup operation following the write to TR6. The hit/miss indication can be found in TR7 bit 4 (see Table 8.5).

TR7 will contain the following information if bit 4 indicated that the lookup test resulted in a hit. Bits 2-3 will indicate in which set the match occurred. The 22 most significant bits in TR7 will contain the physical address and page attributes contained in the entry. Bits 9-7 will contain the LRU bits associated with the accessed set. The state of the LRU bits is previous to their being updated for the current lookup.

If bit 4 in TR7 indicated that the lookup test resulted in a miss the remaining bits in TR7 are undefined.

Again it should be noted that a TLB testability lookup operation affects the state of the LRU bits. The LRU bits will be updated if a hit occurred. The entry which was hit will become the most recently used.

8.4 Tristate Output Test Mode

The 486 microprocessor provides the ability to float all its outputs and bidirectional pins. This includes all pins floated during bus hold as well as pins which are never floated in normal operation of the chip (HLDA, BREQ, FERR# and PCHK#). When the 486 microprocessor is in the tristate output test mode external testing can be used to test board connections.

The tristate test mode is invoked by driving FLUSH# low in the clock prior to RESET going low (see Figure 6.3). The 486 microprocessor remains in the tristate test mode until the next RESET.

9.0 DEBUGGING SUPPORT

The 486 Microprocessor provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

- 1) the code execution breakpoint opcode (0CCH),
- 2) the single-step capability provided by the TF bit in the flag register, and
- 3) the code and data breakpoint capability provided by the Debug Registers DR0–3, DR6, and DR7.

9.1 Breakpoint Instruction

A single-byte-opcode breakpoint instruction is available for use by software debuggers. The breakpoint opcode is 0CCH, and generates an exception 3 trap when executed. In typical use, a debugger program can “plant” the breakpoint instruction at all desired code execution breakpoints. The single-byte breakpoint opcode is an alias for the two-byte general software interrupt instruction, INT n , where $n=3$. The only difference between INT 3 (0CCh) and INT n is that INT 3 is never IOPL-sensitive but INT n is IOPL-sensitive in Protected Mode and Virtual 8086 Mode.

9.2 Single-Step Trap

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1. Precisely, exception 1 occurs as a trap after the instruction following the instruction which set TF. In typical practice, a debugger sets the TF bit of a flag register image on the debugger’s stack. It then typically transfers control to the user program and loads the flag image with a signal instruction, the IRET instruction. The single-step trap occurs after executing one instruction of the user program.

Since the exception 1 occurs as a trap (that is, it occurs after the instruction has already executed), the CS:EIP pushed onto the debugger’s stack points to the next unexecuted instruction of the program being debugged. An exception 1 handler, merely by ending with an IRET instruction, can therefore efficiently support single-stepping through a user program.

9.3 Debug Registers

The Debug Registers are an advanced debugging feature of the 486 Microprocessor. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT3 breakpoint opcode.

The 486 Microprocessor contains six Debug Registers, providing the ability to specify up to four distinct breakpoint addresses, breakpoint control options, and read breakpoint status. Initially after reset, breakpoints are in the disabled state. Therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are autovectored to exception number 1.

9.3.1 LINEAR ADDRESS BREAKPOINT REGISTERS (DR0–DR3)

Up to four breakpoint addresses can be specified by writing into Debug Registers DR0–DR3, shown in Figure 9.1. The breakpoint addresses specified are 32-bit linear addresses. 486 Microprocessor hardware continuously compares the linear breakpoint addresses in DR0–DR3 with the linear addresses generated by executing software (a linear address is the result of computing the effective address and adding the 32-bit segment base address). Note that if paging is not enabled the linear address equals the physical address. If paging is enabled, the linear address is translated to a physical 32-bit address by the on-chip paging unit. Regardless of whether paging is enabled or not, however, the breakpoint registers hold linear addresses.

9.3.2 DEBUG CONTROL REGISTER (DR7)

A Debug Control Register, DR7 shown in Figure 9.1, allows several debug control functions such as enabling the breakpoints and setting up other control options for the breakpoints. The fields within the Debug Control Register, DR7, are as follows:

LEN_{*i*} (breakpoint length specification bits)

A 2-bit LEN field exists for each of the four breakpoints. LEN specifies the length of the associated breakpoint field. The choices for data breakpoints are: 1 byte, 2 bytes, and 4 bytes. Instruction execution breakpoints must have a length of 1 (LEN_{*i*} = 00). Encoding of the LEN_{*i*} field is as follows:

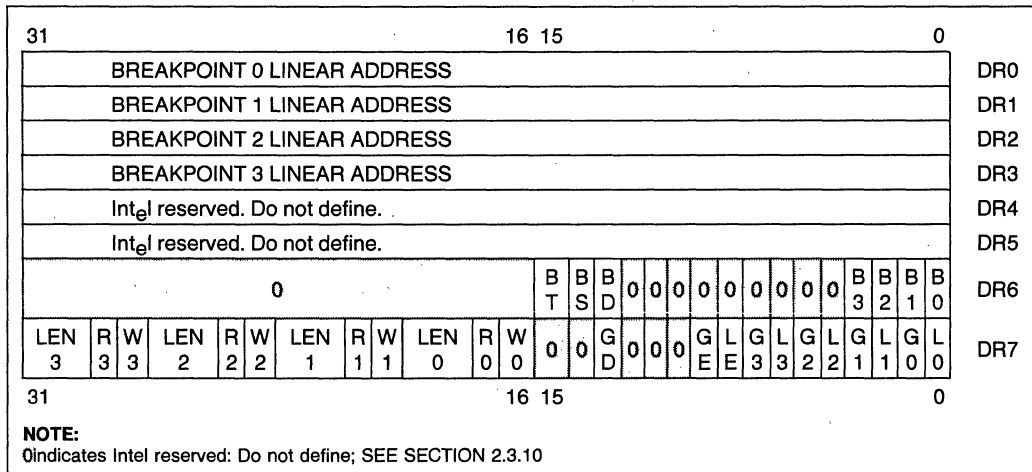
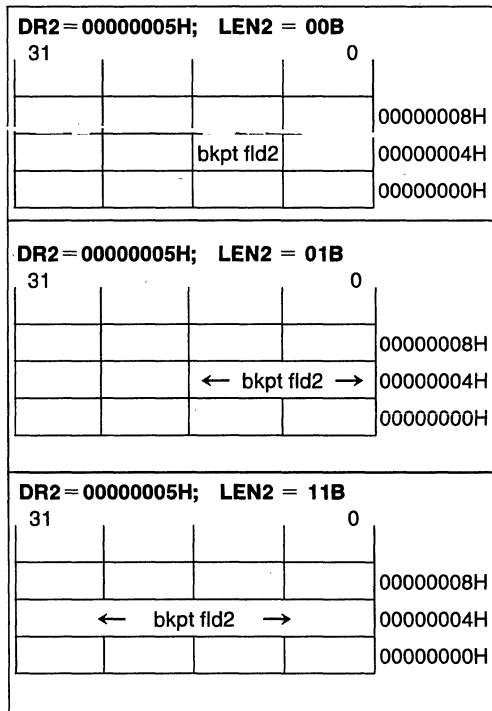


Figure 9.1. Debug Registers

LEN _i Encoding	Breakpoint Field Width	Usage of Least Significant Bits in Breakpoint Address Register i, (i = 0-3)
00	1 byte	All 32-bits used to specify a single-byte breakpoint field.
01	2 bytes	A1-A31 used to specify a two-byte, word-aligned breakpoint field. A0 in Breakpoint Address Register is not used.
10	Undefined—do not use this encoding	
11	4 bytes	A2-A31 used to specify a four-byte, dword-aligned breakpoint field. A0 and A1 in Breakpoint Address Register are not used.

The LEN_i field controls the size of breakpoint field i by controlling whether all low-order linear address bits in the breakpoint address register are used to detect the breakpoint event. Therefore, all breakpoint fields are aligned; 2-byte breakpoint fields begin on Word boundaries, and 4-byte breakpoint fields begin on Dword boundaries.

The following is an example of various size breakpoint fields. Assume the breakpoint linear address in DR2 is 00000005H. In that situation, the following illustration indicates the region of the breakpoint field for lengths of 1, 2, or 4 bytes.



RWi (memory access qualifier bits)

A 2-bit RW field exists for each of the four breakpoints. The 2-bit RW field specifies the type of usage which must occur in order to activate the associated breakpoint.

RW Encoding	Usage Causing Breakpoint
00	Instruction execution only
01	Data writes only
10	Undefined—do not use this encoding
11	Data reads and writes only

RW encoding 00 is used to set up an instruction execution breakpoint. RW encodings 01 or 11 are used to set up write-only or read/write data breakpoints.

Note that **instruction execution breakpoints are taken as faults** (i.e., before the instruction executes), but **data breakpoints are taken as traps** (i.e., after the data transfer takes place).

Using LENi and RWi to Set Data Breakpoint i

A data breakpoint can be set up by writing the linear address into DRi (i = 0–3). For data breakpoints, RWi can = 01 (write-only) or 11 (write/read). LEN can = 00, 01, or 11.

If a data access entirely or partly falls within the data breakpoint field, the data breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 trap will occur.

Using LENi and RWi to Set Instruction Execution Breakpoint i

An instruction execution breakpoint can be set up by writing address of the beginning of the instruction (including prefixes if any) into DRi (i = 0–3). RWi must = 00 and LEN must = 00 for instruction execution breakpoints.

If the instruction beginning at the breakpoint address is about to be executed, the instruction execution breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 fault will occur before the instruction is executed.

Note that an instruction execution breakpoint address must be equal to the **beginning** byte address of an instruction (including prefixes) in order for the instruction execution breakpoint to occur.

GD (Global Debug Register access detect)

The Debug Registers can only be accessed in Real Mode or at privilege level 0 in Protected Mode. The GD bit, when set, provides extra protection against **any** Debug Register access even in Real Mode or at privilege level 0 in Protected Mode. This additional protection feature is provided to guarantee that a software debugger can have full control over the Debug Register resources when required. The GD bit, when set, causes an exception 1 fault if an instruction attempts to read or write any Debug Register. The GD bit is then automatically cleared when the exception 1 handler is invoked, allowing the exception 1 handler free access to the debug registers.

GE and LE (Exact data breakpoint match, global and local)

The breakpoint mechanism of the 486 Microprocessor differs from that of the 386. The 486 Microprocessor always does exact data breakpoint matching, regardless of GE/LE bit settings. Any data breakpoint trap will be reported exactly after completion of the instruction that caused the operand transfer. Exact reporting is provided by forcing the 486 Microprocessor execution unit to wait for completion of data operand transfers before beginning execution of the next instruction.

When the 486 Microprocessor performs a task switch, the LE bit is cleared. Thus, the LE bit supports fast task switching out of tasks, that have enabled the exact data breakpoint match for their task-local breakpoints. The LE bit is cleared by the processor during a task switch, to avoid having exact data breakpoint match enabled in the new task. Note that exact data breakpoint match must be re-enabled under software control.

The 486 Microprocessor GE bit is unaffected during a task switch. The GE bit supports exact data breakpoint match that is to remain enabled during all tasks executing in the system.

Note that **instruction execution** breakpoints are always reported exactly.

Gi and Li (breakpoint enable, global and local)

If either Gi or Li is set then the associated breakpoint (as defined by the linear address in DRi, the length in LENi and the usage criteria in RWi) is enabled. If either Gi or Li is set, and the 486 Microprocessor detects the ith breakpoint condition, then the exception 1 handler is invoked.

When the 486 Microprocessor performs a task switch to a new Task State Segment (TSS), all Li bits are cleared. Thus, the Li bits support fast task switching out of tasks that use some task-local

breakpoint registers. The Li bits are cleared by the processor during a task switch, to avoid spurious exceptions in the new task. Note that the breakpoints must be re-enabled under software control.

All 486 Microprocessor Gi bits are unaffected during a task switch. The Gi bits support breakpoints that are active in all tasks executing in the system.

9.3.3 DEBUG STATUS REGISTER (DR6)

A Debug Status Register, DR6 shown in Figure 9.1, allows the exception 1 handler to easily determine why it was invoked. Note the exception 1 handler can be invoked as a result of one of several events:

- 1) DR0 Breakpoint fault/trap.
- 2) DR1 Breakpoint fault/trap.
- 3) DR2 Breakpoint fault/trap.
- 4) DR3 Breakpoint fault/trap.
- 5) Single-step (TF) trap.
- 6) Task switch trap.
- 7) Fault due to attempted debug register access when GD = 1.

The Debug Status Register contains single-bit flags for each of the possible events invoking exception 1. Note below that some of these events are faults (exception taken before the instruction is executed), while other events are traps (exception taken after the debug events occurred).

The flags in DR6 are set by the hardware but never cleared by hardware. Exception 1 handler software should clear DR6 before returning to the user program to avoid future confusion in identifying the source of exception 1.

The fields within the Debug Status Register, DR6, are as follows:

Bi (debug fault/trap due to breakpoint 0–3)

Four breakpoint indicator flags, B0–B3, correspond one-to-one with the breakpoint registers in DR0–DR3. A flag Bi is set when the condition described by DRI, LENi, and RWi occurs.

If Gi or Li is set, and if the ith breakpoint is detected, the processor will invoke the exception 1 handler. The exception is handled as a fault if an instruction execution breakpoint occurred, or as a trap if a data breakpoint occurred.

IMPORTANT NOTE: A flag Bi is set whenever the hardware detects a match condition on **enabled**

breakpoint i. Whenever a match is detected on at least one **enabled** breakpoint i, the hardware immediately sets all Bi bits corresponding to breakpoint conditions matching at that instant, whether enabled or not. Therefore, the exception 1 handler may see that multiple Bi bits are set, but only set Bi bits corresponding to **enabled** breakpoints (Li or Gi set) are **true** indications of why the exception 1 handler was invoked.

BD (debug fault due to attempted register access when GD bit set)

This bit is set if the exception 1 handler was invoked due to an instruction attempting to read or write to the debug registers when GD bit was set. If such an event occurs, then the GD bit is automatically cleared when the exception 1 handler is invoked, allowing handler access to the debug registers.

BS (debug trap due to single-step)

This bit is set if the exception 1 handler was invoked due to the TF bit in the flag register being set (for single-stepping).

BT (debug trap due to task switch)

This bit is set if the exception 1 handler was invoked due to a task switch occurring to a task having a 486 Microprocessor TSS with the T bit set. Note the task switch into the new task occurs normally, but before the first instruction of the task is executed, the exception 1 handler is invoked. With respect to the task switch operation, the operation is considered to be a trap.

9.3.4 USE OF RESUME FLAG (RF) IN FLAG REGISTER

The Resume Flag (RF) in the flag word can suppress an instruction execution breakpoint when the exception 1 handler returns to a user program at a user address which is also an instruction execution breakpoint.

10.0 INSTRUCTION SET SUMMARY

This section describes the 486 microprocessor instruction set. Tables 10.1 through 10.3 list all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in Section 10.2, which completely describes the encoding structure and the definition of all fields occurring within the 486 microprocessor instructions.

10.1 i486™ Microprocessor Instruction Encoding and Clock Count Summary

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Tables 10.1 through 10.3 by the processor clock period (e.g., 40 ns for a 25 MHz 486 microprocessor).

For more detailed information on the encodings of instructions, refer to Section 10.2 Instruction Encodings. Section 10.2 explains the general structure of instruction encodings, and defines exactly the encodings of all fields contained within the instruction.

INSTRUCTION CLOCK COUNT ASSUMPTIONS

The 486 microprocessor instruction clock count tables give clock counts assuming data and instruction accesses hit in the cache. A separate penalty column defines clocks to add if a data access misses in the cache. The combined instruction and data cache hit rate is over 90%.

A cache miss will force the 486 microprocessor to run an external bus cycle. The 486 microprocessor 32-bit burst bus is defined as $r-b-w$.

Where:

r = The number of clocks in the first cycle of a burst read or the number of clocks per data cycle in a non-burst read.

b = The number of clocks for the second and subsequent cycles in a burst read.

w = The number of clocks for a write.

The fastest bus the 486 microprocessor can support is $2-1-2$ assuming 0 wait states. The clock counts in the cache miss penalty column assume a $2-1-2$ bus. For slower busses add $r-2$ clocks to the cache miss penalty for the first dword accessed. Other factors also affect instruction clock counts.

Instruction Clock Count Assumptions

- The external bus is available for reads or writes at all times. Else add clocks to reads until the bus is available.
- Accesses are aligned. Add three clocks to each misaligned access.
- Cache fills complete before subsequent accesses to the same line. If a read misses the cache during a cache fill due to a previous read or pre-fetch, the read must wait for the cache fill to complete. If a read or write accesses a cache line still being filled, it must wait for the fill to complete.
- If an effective address is calculated, the base register is not the destination register of the preceding instruction. If the base register is the destination register of the preceding instruction add 1 to the clock counts shown. Back-to-back PUSH and POP instructions are not affected by this rule.
- An effective address calculation uses one base register and does not use an index register. However, if the effective address calculation uses an index register, 1 clock **may** be added to the clock count shown.
- The target of a jump is in the cache. If not, add r clocks for accessing the destination instruction of a jump. If the destination instruction is not completely contained in the first dword read, add a maximum of $3b$ clocks. If the destination instruction is not completely contained in the first 16 byte burst, add a maximum of another $r+3b$ clocks.
- If no write buffer delay, w clocks are added only in the case in which all write buffers are full. Typically, this case rarely occurs.
- Displacement and immediate not used together. If displacement and immediate used together, 1 clock **may** be added to the clock count shown.
- No invalidate cycles. Add a delay of 1 clock for each invalidate cycle if the invalidate cycle contends for the internal cache/external bus when the 486 CPU needs to use it.
- Page translation hits in TLB. A TLB miss will add 13, 21 or 28 clocks to the instruction depending on whether the Accessed and/or Dirty bit in neither, one or both of the page entries needs to be set in memory. This assumes that neither page entry is in the data cache and a page fault does not occur on the address translation.
- No exceptions are detected during instruction execution. Refer to Interrupt Clock Counts Table for extra clocks if an interrupt is detected.
- Instructions that read multiple consecutive data items (i.e. task switch, POPA, etc.) and miss the cache are assumed to start the first access on a 16-byte boundary. If not, an extra cache line fill may be necessary which may add up to $(r+3b)$ clocks to the cache miss penalty.

Table 10.1. i486™ Microprocessor Integer Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS				
MOV = Move:				
reg1 to reg2	1000100W 11 reg1 reg2	1		
reg2 to reg1	1000101w 11 reg1 reg2	1		
memory to reg	1000101w mod reg r/m	1	2	
reg to memory	1000100w mod reg r/m	1		
Immediate to reg	1100011w 11000 reg immediate data	1		
or	1011w reg immediate data	1		
Immediate to Memory	1100011w mod 000 r/m displacement immediate	1		
Memory to Accumulator	1010000w full displacement	1	2	
Accumulator to Memory	1010001w full displacement	1		
MOVX/MOVZX = Move with Sign/Zero Extension				
reg2 to reg1	00001111 1011z11w 11 reg1 reg2	3		
memory to reg	00001111 1011z11w mod reg r/m	3	2	
z Instruction				
0	MOVZX			
1	MOVXS			
PUSH = Push				
reg	11111111 11 110 reg	4		
or	01010 reg	1		
memory	11111111 mod 110 r/m	4	1	1
immediate	011010s0 immediate data	1		
PUSHA = Push All	01100000	11		
POP = Pop				
reg	10001111 11 000 reg	4	1	
or	01011 reg	1	2	
memory	10001111 mod 000 r/m	5	2	1
POPA = Pop All	01100001	9	7/15	16/32
XCHG = Exchange				
reg1 with reg2	1000011w 11 reg1 reg2	3		2
Accumulator with reg	10010 reg	3		2
Memory with reg	1000011w mod reg r/m	5		2
NOP = No Operation	10010000	1		
LEA = Load EA to Register				
no index register	10001101 mod reg r/m	1		
with index register		2		



Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
Instruction	TTT			
ADD = Add	000			
ADC = Add with Carry	010			
AND = Logical AND	100			
OR = Logical OR	001			
SUB = Subtract	101			
SBB = Subtract with Borrow	011			
XOR = Logical Exclusive OR	110			
reg1 to reg2	00TTT00w 11 reg1 reg2	1		
reg2 to reg1	00TTT01w 11 reg1 reg2	1		
memory to register	00TTT01w mod reg r/m	2	2	
register to memory	00TTT00w mod reg r/m	3	6/2	U/L
immediate to register	100000sw 11 TTT reg immediate register	1		
immediate to accumulator	00TTT10w immediate data	1		
immediate to memory	100000sw mod TTT r/m immediate data	3	6/2	U/L
Instruction	TTT			
INC = Increment	000			
DEC = Decrement	001			
reg	1111111w 11 TTT reg	1		
or	01TTT reg	1		
memory	1111111w mod TTT r/m	3	6/2	U/L
Instruction	TTT			
NOT = Logical Complement	010			
NEG = Negate	011			
reg	1111011w 11 TTT reg	1		
memory	1111011w mod TTT r/m	3	6/2	U/L
CMP = Compare				
reg1 with reg2	0011100w 11 reg1 reg2	1		
reg2 with reg1	0011101w 11 reg1 reg2	1		
memory with register	0011100w mod reg r/m	2	2	
register with memory	0011101w mod reg r/m	2	2	
immediate with register	100000sw 11 111 reg immediate data	1		
immediate with acc.	0011110w immediate data	1		
immediate with memory	100000sw mod 111 r/m immediate data	2	2	
TEST = Logical Compare				
reg1 and reg2	1000010w 11 reg1 reg2	1		
memory and register	1000010w mod reg r/m	2	2	
immediate and register	1111011w 11 000 reg immediate data	1		
immediate and acc.	1010100w immediate data	1		
immediate and memory	1111011w mod 000 r/m immediate data	2	2	

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
MUL = Multiply (unsigned)				
acc. with register	1111011w 11 100 reg			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
acc. with memory	1111011w mod 100 r/m			
Multiplier-Byte		13/18	1	MN/MX, 3
Word		13/26	1	MN/MX, 3
Dword		13/42	1	MN/MX, 3
IMUL = Integer Multiply (signed)				
acc. with register	1111011w 11 101 reg			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
acc. with memory	1111011w mod 101 r/m			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
reg1 with reg2	00001111 10101111 11 reg1 reg2			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
register with memory	00001111 10101111 mod reg r/m			
Multiplier-Byte		13/18	1	MN/MX, 3
Word		13/26	1	MN/MX, 3
Dword		13/42	1	MN/MX, 3
reg1 with imm. to reg2	011010s1 11 reg1 reg2 immediate data			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
mem. with imm. to reg.	011010s1 mod reg r/m immediate data			
Multiplier-Byte		13/18	2	MN/MX, 3
Word		13/26	2	MN/MX, 3
Dword		13/42	2	MN/MX, 3
DIV = Divide (unsigned)				
acc. by register	1111011w 11 110 reg			
Divisor-Byte				
Word		16		
Dword		24		
acc. by memory	1111011w mod 110 r/m			
Divisor-Byte				
Word		16		
Dword		24		
		40		
IDIV = Integer Divide (signed)				
acc. by register	1111011w 11 111 reg			
Divisor-Byte				
Word		19		
Dword		27		
		43		



Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes																
INTEGER OPERATIONS (Continued)																				
acc. by memory	1111011w mod 111 r/m																			
Divisor-Byte		20																		
Word		28																		
Dword		44																		
CBW = Convert Byte to Word	10011000	3																		
CWD = Convert Word to Dword	10011001	3																		
<table border="1"> <thead> <tr> <th>Instruction</th> <th>TTT</th> </tr> </thead> <tbody> <tr> <td>ROL = Rotate Left</td> <td>000</td> </tr> <tr> <td>ROR = Rotate Right</td> <td>001</td> </tr> <tr> <td>RCL = Rotate through Carry Left</td> <td>010</td> </tr> <tr> <td>RCR = Rotate through Carry Right</td> <td>011</td> </tr> <tr> <td>SHL/SAL = Shift Logical/Arithmetic Left</td> <td>100</td> </tr> <tr> <td>SHR = Shift Logical Right</td> <td>101</td> </tr> <tr> <td>SAR = Shift Arithmetic Right</td> <td>111</td> </tr> </tbody> </table>					Instruction	TTT	ROL = Rotate Left	000	ROR = Rotate Right	001	RCL = Rotate through Carry Left	010	RCR = Rotate through Carry Right	011	SHL/SAL = Shift Logical/Arithmetic Left	100	SHR = Shift Logical Right	101	SAR = Shift Arithmetic Right	111
Instruction	TTT																			
ROL = Rotate Left	000																			
ROR = Rotate Right	001																			
RCL = Rotate through Carry Left	010																			
RCR = Rotate through Carry Right	011																			
SHL/SAL = Shift Logical/Arithmetic Left	100																			
SHR = Shift Logical Right	101																			
SAR = Shift Arithmetic Right	111																			
Not Through Carry (ROL, ROR, SAL, SAR, SHL, and SHR)																				
reg by 1	1101000w 11 TTT reg	3																		
memory by 1	1101000w mod TTT r/m	4	6																	
reg by CL	1101001w 11 TTT reg	3																		
memory by CL	1101001w mod TTT r/m	4	6																	
reg by immediate count	1100000w 11 TTT reg	2		immediate 8-bit data																
mem by immediate count	1100000w mod TTT r/m	4	6	immediate 8-bit data																
Through Carry (RCL and RCR)																				
reg by 1	1101000w 11 TTT reg	3																		
memory by 1	1101000w mod TTT r/m	4	6																	
reg by CL	1101001w 11 TTT reg	8/30		MN/MX, 4																
memory by CL	1101001w mod TTT r/m	9/31		MN/MX, 5																
reg by immediate count	1100000w 11 TTT reg	8/30		immediate 8-bit data MN/MX, 4																
mem by immediate count	1100000w mod TTT r/m	9/31		immediate 8-bit data MN/MX, 5																
<table border="1"> <thead> <tr> <th>Instruction</th> <th>TTT</th> </tr> </thead> <tbody> <tr> <td>SHLD = Shift Left Double</td> <td>100</td> </tr> <tr> <td>SHRD = Shift Right Double</td> <td>101</td> </tr> </tbody> </table>					Instruction	TTT	SHLD = Shift Left Double	100	SHRD = Shift Right Double	101										
Instruction	TTT																			
SHLD = Shift Left Double	100																			
SHRD = Shift Right Double	101																			
register with immediate	00001111 10TTT100 11 reg2 reg1	2		imm 8-bit data																
memory by immediate	00001111 10TTT100 mod reg r/m	3	6	imm 8-bit data																
register by CL	00001111 10TTT101 11 reg2 reg1	3																		
memory by CL	00001111 10TTT101 mod reg r/m	4	5																	
BSWAP = Byte Swap	00001111 11001 reg	1																		
XADD = Exchange and Add																				
reg1, reg2	00001111 1100000w 11 reg2 reg1	3																		
memory, reg	00001111 1100000w mod reg r/m	4	6/2	U/L																
CMPXCHG = Compare and Exchange																				
reg1, reg2	00001111 1011000w 11 reg2 reg1	6																		
memory, reg	00001111 1011000w mod reg r/m	7/10	2	6																

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
CONTROL TRANSFER (within segment)				
NOTE: Times are jump taken/not taken				
Jccc = Jump on cccc				
8-bit displacement	0 1 1 1 t t t n 8-bit disp.	3/1		T/NT, 23
full displacement	0 0 0 0 1 1 1 1 1 0 0 0 t t t n full displacement	3/1		T/NT, 23
NOTE: Times are jump taken/not taken				
SETccccc = Set Byte on ccccc (Times are ccccc true/false)				
reg	0 0 0 0 1 1 1 1 1 0 0 1 t t t n 1 1 0 0 0 reg	4/3		
memory	0 0 0 0 1 1 1 1 1 0 0 1 t t t n mod 0 0 0 r/m	3/4		
Mnemonic cccc	Condition	t t t n		
O	Overflow	0000		
NO	No Overflow	0001		
B/NAE	Below/Not Above or Equal	0010		
NB/AE	Not Below/Above or Equal	0011		
E/Z	Equal/Zero	0100		
NE/NZ	Not Equal/Not Zero	0101		
BE/NA	Below or Equal/Not Above	0110		
NBE/A	Not Below or Equal/Above	0111		
S	Sign	1000		
NS	Not Sign	1001		
P/PE	Parity/Parity Even	1010		
NP/PO	Not Parity/Parity Odd	1011		
L/NGE	Less Than/Not Greater or Equal	1100		
NL/GE	Not Less Than/Greater or Equal	1101		
LE/NG	Less Than or Equal/Greater Than	1110		
NLE/G	Not Less Than or Equal/Greater Than	1111		
LOOP = LOOP CX Times	1 1 1 0 0 0 1 0 8-bit disp.	7/6		L/NL, 23
LOOPZ/LOOPE = Loop with Zero/Equal	1 1 1 0 0 0 0 1 8-bit disp.	9/6		L/NL, 23
LOOPNZ/LOOPNE = Loop while Not Zero	1 1 1 0 0 0 0 0 8-bit disp.	9/6		L/NL, 23
JCXZ = Jump on CX Zero	1 1 1 0 0 0 1 1 8-bit disp.	8/5		T/NT, 23
JECXZ = Jump on ECX Zero	1 1 1 0 0 0 1 1 8-bit disp.	8/5		T/NT, 23
(Address Size Prefix Differentiates JCXZ for JECXZ)				
JMP = Unconditional Jump (within segment)				
Short	1 1 1 0 1 0 1 1 8-bit disp.	3		7, 23
Direct	1 1 1 0 1 0 0 1 full displacement	3		7, 23
Register Indirect	1 1 1 1 1 1 1 1 1 1 1 0 0 reg	5		7, 23
Memory Indirect	1 1 1 1 1 1 1 1 mod 1 0 0 r/m	5	5	7
CALL = Call (within segment)				
Direct	1 1 1 0 1 0 0 0 full displacement	3		7, 23
Register Indirect	1 1 1 1 1 1 1 1 1 1 0 1 0 reg	5		7, 23
Memory Indirect	1 1 1 1 1 1 1 1 mod 0 1 0 r/m	5	5	7
RET = Return from CALL (within segment)				
	1 1 0 0 0 0 1 1	5	5	
Adding Immediate to SP	1 1 0 0 0 0 1 0 16-bit disp.	5	5	

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty If Cache Miss	Notes			
CONTROL TRANSFER (within segment) (Continued)							
ENTER = Enter Procedure	<table border="1"><tr><td>11001000</td><td>16-bit disp., 8-bit level</td></tr></table>	11001000	16-bit disp., 8-bit level				
11001000	16-bit disp., 8-bit level						
Level = 0		14					
Level = 1		17					
Level (L) > 1		17+3L		8			
LEAVE = Leave Procedure	<table border="1"><tr><td>11001001</td></tr></table>	11001001	5	1			
11001001							
MULTIPLE-SEGMENT INSTRUCTIONS							
MOV = Move							
reg. to segment reg.	<table border="1"><tr><td>10001110</td><td>11 sreg3 reg</td></tr></table>	10001110	11 sreg3 reg	3/9	0/3	RV/P, 9	
10001110	11 sreg3 reg						
memory to segment reg.	<table border="1"><tr><td>10001110</td><td>mod sreg3 r/m</td></tr></table>	10001110	mod sreg3 r/m	3/9	2/5	RV/P, 9	
10001110	mod sreg3 r/m						
segment reg. to reg.	<table border="1"><tr><td>10001100</td><td>11 sreg3 reg</td></tr></table>	10001100	11 sreg3 reg	3			
10001100	11 sreg3 reg						
segment reg. to memory	<table border="1"><tr><td>10001100</td><td>mod sreg3 r/m</td></tr></table>	10001100	mod sreg3 r/m	3			
10001100	mod sreg3 r/m						
PUSH = Push							
segment reg. (ES, CS, SS, or DS)	<table border="1"><tr><td>000sreg2110</td></tr></table>	000sreg2110	3				
000sreg2110							
segment reg. (FS or GS)	<table border="1"><tr><td>00001111</td><td>10 sreg3000</td></tr></table>	00001111	10 sreg3000	3			
00001111	10 sreg3000						
POP = Pop							
segment reg. (ES, SS, or DS)	<table border="1"><tr><td>000sreg2111</td></tr></table>	000sreg2111	3/9	2/5	RV/P, 9		
000sreg2111							
segment reg. (FS or GS)	<table border="1"><tr><td>00001111</td><td>10 sreg3001</td></tr></table>	00001111	10 sreg3001	3/9	2/5	RV/P, 9	
00001111	10 sreg3001						
LDS = Load Pointer to DS	<table border="1"><tr><td>11000101</td><td>mod reg r/m</td></tr></table>	11000101	mod reg r/m	6/12	7/10	RV/P, 9	
11000101	mod reg r/m						
LES = Load Pointer to ES	<table border="1"><tr><td>11000100</td><td>mod reg r/m</td></tr></table>	11000100	mod reg r/m	6/12	7/10	RV/P, 9	
11000100	mod reg r/m						
LFS = Load Pointer to FS	<table border="1"><tr><td>00001111</td><td>10110100</td><td>mod reg r/m</td></tr></table>	00001111	10110100	mod reg r/m	6/12	7/10	RV/P, 9
00001111	10110100	mod reg r/m					
LGS = Load Pointer to GS	<table border="1"><tr><td>00001111</td><td>10110101</td><td>mod reg r/m</td></tr></table>	00001111	10110101	mod reg r/m	6/12	7/10	RV/P, 9
00001111	10110101	mod reg r/m					
LSS = Load Pointer to SS	<table border="1"><tr><td>00001111</td><td>10110010</td><td>mod reg r/m</td></tr></table>	00001111	10110010	mod reg r/m	6/12	7/10	RV/P, 9
00001111	10110010	mod reg r/m					
CALL = Call							
Direct intersegment	<table border="1"><tr><td>10011010</td><td>unsigned full offset, selector</td></tr></table>	10011010	unsigned full offset, selector	18	2	R, 7, 22	
10011010	unsigned full offset, selector						
to same level		20	3	P, 9			
thru Gate to same level		35	6	P, 9			
to inner level, no parameters		69	17	P, 9			
to inner level, x parameter (d) words		77+4X	17+n	P, 11, 9			
to TSS		37+TS	3	P, 10, 9			
thru Task Gate		38+TS	3	P, 10, 9			
Indirect intersegment	<table border="1"><tr><td>11111111</td><td>mod 011 r/m</td></tr></table>	11111111	mod 011 r/m	17	8	R, 7	
11111111	mod 011 r/m						
to same level		20	10	P, 9			
thru Gate to same level		35	13	P, 9			
to inner level, no parameters		69	24	P, 9			
to inner level, x parameter (d) words		77+4X	24+n	P, 11, 9			
to TSS		37+TS	10	P, 10, 9			
thru Task Gate		38+TS	10	P, 10, 9			
RET = Return from CALL							
intersegment	<table border="1"><tr><td>11001011</td></tr></table>	11001011	13	8	R, 7		
11001011							
to same level		17	9	P, 9			
to outer level		35	12	P, 9			
intersegment adding	<table border="1"><tr><td>11001010</td><td>16-bit disp.</td></tr></table>	11001010	16-bit disp.				
11001010	16-bit disp.						
imm. to SP		14	8	R, 7			
to same level		18	9	P, 9			
to outer level		36	12	P, 9			

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes								
MULTIPLE-SEGMENT INSTRUCTIONS (Continued)												
JMP = Unconditional Jump												
Direct intersegment	11101010 unsigned full offset, selector	17	2	R, 7, 22								
to same level		19	3	P, 9								
thru Call Gate to same level		32	6	P, 9								
thru TSS		42+TS	3	P, 10, 9								
thru Task Gate		43+TS	3	P, 10, 9								
Indirect intersegment	11111111 mod 101 r/m	13	9	R, 7, 9								
to same level		18	10	P, 9								
thru Call Gate to same level		31	13	P, 9								
thru TSS		41+TS	10	P, 10, 9								
thru Task Gate		42+TS	10	P, 10, 9								
BIT MANIPULATION												
BT = Test bit												
register, immediate	00001111 10111010 11 100 reg	imm. 8-bit data	3									
memory, immediate	00001111 10111010 mod 100 r/m	imm. 8-bit data	3	1								
reg1, reg2	00001111 10100011 11 reg2 reg1		3									
memory, reg	00001111 10100011 mod reg r/m		8	2								
<table border="1"> <thead> <tr> <th>Instruction</th> <th>TTT</th> </tr> </thead> <tbody> <tr> <td>BTS = Test Bit and Set</td> <td>101</td> </tr> <tr> <td>BTR = Test Bit and Reset</td> <td>110</td> </tr> <tr> <td>BTC = Test Bit and Compliment</td> <td>111</td> </tr> </tbody> </table>					Instruction	TTT	BTS = Test Bit and Set	101	BTR = Test Bit and Reset	110	BTC = Test Bit and Compliment	111
Instruction	TTT											
BTS = Test Bit and Set	101											
BTR = Test Bit and Reset	110											
BTC = Test Bit and Compliment	111											
register, immediate	00001111 10111010 11 TTT reg	imm. 8-bit data	6									
memory, immediate	00001111 10111010 mod TTT r/m	imm. 8-bit data	8	2/0 U/L								
reg1, reg2	00001111 10TTT011 11 reg2 reg1		6									
memory, reg	00001111 10TTT011 mod reg r/m		13	3/1 U/L								
BSF = Scan Bit Forward												
reg1, reg2	00001111 10111100 11 reg2 reg1		6/42	MN/MX, 12								
memory, reg	00001111 10111100 mod reg r/m		7/43	2 MN/MX, 13								
BSR = Scan Bit Reverse												
reg1, reg2	00001111 10111101 11 reg2 reg1		6/103	MN/MX, 14								
memory, reg	00001111 10111101 mod reg r/m		7/104	1 MN/MX, 15								
STRING INSTRUCTIONS												
CMPS = Compare Byte/Word	1010011w	8	6	16								
LODS = Load Byte/Word to AL/AX/EAX	1010110w	5	2									
MOVS = Move Byte/Word	1010010w	7	2	16								
SCAS = Scan Byte/Word	1010111w	6	2									
STOS = Store Byte/Word from AL/AX/EX	1010101w	5										
XLAT = Translate String	11010111	4	2									

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes		
REPEATED STRING INSTRUCTIONS						
Repeated by Count in CX or ECX (C = Count in CX or ECX)						
REPE CMPS = Compare String (Find Non-Match) C = 0 C > 0	<table border="1"><tr><td>11110011</td><td>1010011w</td></tr></table>	11110011	1010011w	5 7+7c		16, 17
11110011	1010011w					
REPNE CMPS = Compare String (Find Match) C = 0 C > 0	<table border="1"><tr><td>11110010</td><td>1010011w</td></tr></table>	11110010	1010011w	5 7+7c		16, 17
11110010	1010011w					
REP LODS = Load String C = 0 C > 0	<table border="1"><tr><td>11110010</td><td>1010110w</td></tr></table>	11110010	1010110w	5 7+4c		16, 18
11110010	1010110w					
REP MOVS = Move String C = 0 C = 1 C > 1	<table border="1"><tr><td>11110010</td><td>1010010w</td></tr></table>	11110010	1010010w	5 13 12+3c	1	16 16, 19
11110010	1010010w					
REPE SCAS = Scan String (Find Non-AL/AX/EAX) C = 0 C > 0	<table border="1"><tr><td>11110011</td><td>1010111w</td></tr></table>	11110011	1010111w	5 7+5c		20
11110011	1010111w					
REPNE SCAS = Scan String (Find AL/AX/EAX) C = 0 C > 0	<table border="1"><tr><td>11110010</td><td>1010111w</td></tr></table>	11110010	1010111w	5 7+5c		20
11110010	1010111w					
REP STOS = Store String C = 0 C > 0	<table border="1"><tr><td>11110010</td><td>1010101w</td></tr></table>	11110010	1010101w	5 7+4c		
11110010	1010101w					
FLAG CONTROL						
CLC = Clear Carry Flag	<table border="1"><tr><td>11111000</td></tr></table>	11111000	2			
11111000						
STC = Set Carry Flag	<table border="1"><tr><td>11111001</td></tr></table>	11111001	2			
11111001						
CMC = Complement Carry Flag	<table border="1"><tr><td>11110101</td></tr></table>	11110101	2			
11110101						
CLD = Clear Direction Flag	<table border="1"><tr><td>11111100</td></tr></table>	11111100	2			
11111100						
STD = Set Direction Flag	<table border="1"><tr><td>11111101</td></tr></table>	11111101	2			
11111101						
CLI = Clear Interrupt Enable Flag	<table border="1"><tr><td>11111010</td></tr></table>	11111010	5			
11111010						
STI = Set Interrupt Enable Flag	<table border="1"><tr><td>11111011</td></tr></table>	11111011	5			
11111011						
LAHF = Load AH into Flag	<table border="1"><tr><td>10011111</td></tr></table>	10011111	3			
10011111						
SAHF = Store AH into Flags	<table border="1"><tr><td>10011110</td></tr></table>	10011110	2			
10011110						
PUSHF = Push Flags	<table border="1"><tr><td>10011100</td></tr></table>	10011100	4/3		RV/P	
10011100						
POPF = Pop Flags	<table border="1"><tr><td>10011101</td></tr></table>	10011101	9/6		RV/P	
10011101						
DECIMAL ARITHMETIC						
AAA = ASCII Adjust for Add	<table border="1"><tr><td>00110111</td></tr></table>	00110111	3			
00110111						
AAS = ASCII Adjust for Subtract	<table border="1"><tr><td>00111111</td></tr></table>	00111111	3			
00111111						
AAM = ASCII Adjust for Multiply	<table border="1"><tr><td>11010100</td><td>00001010</td></tr></table>	11010100	00001010	15		
11010100	00001010					

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
DECIMAL ARITHMETIC (Continued)				
AAD = ASCII Adjust for Divide	11010101 00001010	14		
DAA = Decimal Adjust for Add	00100111	2		
DAS = Decimal Adjust for Subtract	00101111	2		
PROCESSOR CONTROL INSTRUCTIONS				
HLT = Halt	11110100	4		
MOV = Move To and From Control/Debug/Test Registers				
CR0 from register	00001111 00100010 11 000 reg	17	2	
CR2/CR3 from register	00001111 00100010 11 eee reg	4		
Reg from CR0-3	00001111 00100000 11 eee reg	4		
DR0-3 from register	00001111 00100011 11 eee reg	10		
DR6-7 from register	00001111 00100011 11 eee reg	10		
Register from DR6-7	00001111 00100001 11 eee reg	9		
Register from DR0-3	00001111 00100001 11 eee reg	9		
TR3 from register	00001111 00100110 11 011 reg	4		
TR4-7 from register	00001111 00100110 11 eee reg	4		
Register from TR3	00001111 00100100 11 011 reg	3		
Register from TR4-7	00001111 00100100 11 eee reg	4		
CLTS = Clear Task Switched Flag	00001111 00000110	7	2	
INVD = Invalidate Data Cache	00001111 00001000	4		
WBINVD = Write-Back and Invalidate Data Cache	00001111 00001001	5		
INVLPG = Invalidate TLB Entry				
INVLPG memory	00001111 00000001 mod 111 r/m	12/11		H/NH
PREFIX BYTES				
Address Size Prefix	01100111	1		
LOCK = Bus Lock Prefix	11110000	1		
Operand Size Prefix	01100110	1		
Segment Override Prefix				
CS:	00101110	1		
DS:	00111110	1		
ES:	00100110	1		
FS:	01100100	1		
GS:	01100101	1		
SS:	00110110	1		

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty If Cache Miss	Notes
PROTECTION CONTROL				
ARPL = Adjust Requested Privilege Level				
From register	0 1 1 0 0 0 1 1 1 1 reg1 reg2	9		
From memory	0 1 1 0 0 0 1 1 mod reg r/m	9		
LAR = Load Access Rights				
From register	0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 0 1 1 reg1 reg2	11	3	
From memory	0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 0 mod reg r/m	11	5	
LGDT = Load Global Descriptor				
Table register	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 mod 0 1 0 r/m	12	5	
LIDT = Load Interrupt Descriptor				
Table register	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 mod 0 1 1 r/m	12	5	
LLDT = Load Local Descriptor				
Table register from reg.	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 1 0 reg	11	3	
Table register from mem.	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 mod 0 1 0 r/m	11	6	
LMSW = Load Machine Status Word				
From register	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 0 reg	13		
From memory	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 mod 1 1 0 r/m	13	1	
LSL = Load Segment Limit				
From register	0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1 reg1 reg2	10	3	
From memory	0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 mod reg r/m	10	6	
LTR = Load Task Register				
From Register	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 0 1 reg	20		
From Memory	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 mod 0 0 1 r/m	20		
SGDT = Store Global Descriptor Table				
	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 mod 0 0 0 r/m	10		
SIDT = Store Interrupt Descriptor Table				
	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 mod 0 0 1 r/m	10		
SLDT = Store Local Descriptor Table				
To register	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 reg	2		
To memory	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 mod 0 0 0 r/m	3		
SMSW = Store Machine Status Word				
To register	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 reg	2		
To memory	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 mod 1 0 0 r/m	3		
STR = Store Task Register				
To register	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 0 1 reg	2		
To memory	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 mod 0 0 1 r/m	3		
VERR = Verify Read Access				
Register	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 r/m	11	3	
Memory	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 mod 1 0 0 r/m	11	7	
VERW = Verify Write Access				
To register	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 1 reg	11	3	
To memory	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 mod 1 0 1 r/m	11	7	

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTERRUPT INSTRUCTIONS				
INT n = Interrupt Type n	1 1 0 0 1 1 0 1 type	INT+4/0		RV/P, 21
INT 3 = Interrupt Type 3	1 1 0 0 1 1 0 0	INT+0		21
INTO = Interrupt 4 If Overflow Flag Set Taken Not Taken	1 1 0 0 1 1 1 0	INT+2		21
		3		21
BOUND = Interrupt 5 If Detect Value Out Range If in range If out of range	0 1 1 0 0 0 1 0 mod reg r/m	7	7	21
		INT+24	7	21
IRET = Interrupt Return Real Mode/Virtual Mode Protected Mode To same level To outer level To nested task (EFLAGS.NT = 1)	1 1 0 0 1 1 1 1	15	8	
		20	11	9
		36	19	9
		TS+32	4	9, 10
External Interrupt		INT+11		21
NMI = Non-Maskable Interrupt		INT+3		21
Page Fault		INT+24		21
VM86 Exceptions				
CLI		INT+8		21
STI		INT+8		21
INT n		INT+9		
PUSHF		INT+9		21
POPF		INT+8		21
IRET		INT+9		
IN				
Fixed Port		INT+50		21
Variable Port		INT+51		21
OUT				
Fixed Port		INT+50		21
Variable Port		INT+51		21
INS		INT+50		21
OUTS		INT+50		21
REP INS		INT+51		21
REP OUTS		INT+51		21

Task Switch Clock Counts Table		
Method	Value for TS	
	Cache Hit	Miss Penalty
VM/486 CPU/286 TSS To 486 CPU TSS	162	55
VM/486 CPU/286 TSS To 286 TSS	143	31
VM/486 CPU/286 TSS To VM TSS	140	37

Interrupt Clock Counts Table			
Method	Value for INT		
	Cache Hit	Miss Penalty	Notes
Real Mode	26	2	
Protected Mode			
Interrupt/Trap gate, same level	44	6	9
Interrupt/Trap gate, different level	71	17	9
Task Gate	37 + TS	3	9, 10
Virtual Mode			
Interrupt/Trap gate, different level	82	17	
Task gate	37 + TS	3	10

Abbreviations	Definition
16/32	16/32 bit modes
U/L	unlocked/locked
MN/MX	minimum/maximum
L/NL	loop/no loop
RV/P	real and virtual mode/protected mode
R	real mode
P	protected mode
T/NT	taken/not taken
H/NH	hit/no hit

NOTES:

- Assuming that the operand address and stack address fall in different cache sets.
- Always locked, no cache hit case.
- Clocks = $10 + \max(\log_2(m), n)$
 m = multiplier value (min clocks for $m=0$)
 n = $3/5$ for $\pm m$
- Clocks = $\{\text{quotient}(\text{count}/\text{operand length})\} * 7 + 9$
 = 8 if count \leq operand length (8/16/32)
- Clocks = $\{\text{quotient}(\text{count}/\text{operand length})\} * 7 + 9$
 = 9 if count \leq operand length (8/16/32)
- Equal/not equal cases (penalty is the same regardless of lock).
- Assuming that addresses for memory read (for indirection), stack push/pop, and branch fall in different cache sets.
- Penalty for cache miss: add 6 clocks for every 16 bytes copied to new stack frame.
- Add 11 clocks for each unaccessed descriptor load.
- Refer to task switch clock counts table for value of TS.
- Add 4 extra clocks to the cache miss penalty for each 16 bytes.
- For notes 12–13: (b = 0–3, non-zero byte number);
 (i = 0–1, non-zero nibble number);
 (n = 0–3, non bit number in nibble);
- Clocks = $8 + 4(b+1) + 3(i+1) + 3(n+1)$
 = 6 if second operand = 0
- Clocks = $9 + 4(b+1) + 3(i+1) + 3(n+1)$
 = 7 if second operand = 0
- For notes 14–15: (n = bit position 0–31)
- Clocks = $7 + 3(32-n)$
 6 if second operand = 0
- Clocks = $8 + 3(32-n)$
 7 if second operand = 0
- Assuming that the two string addresses fall in different cache sets.
- Cache miss penalty: add 6 clocks for every 16 bytes compared. Entire penalty on first compare.
- Cache miss penalty: add 2 clocks for every 16 bytes of data. Entire penalty on first load.
- Cache miss penalty: add 4 clocks for every 16 bytes moved.
 (1 clock for the first operation and 3 for the second)
- Cache miss penalty: add 4 clocks for every 16 bytes scanned.
 (2 clocks each for first and second operations)
- Refer to interrupt clock counts table for value of INT
- Clock count includes one clock for using both displacement and immediate.
- Refer to assumption 6 in the case of a cache miss.

Table 10.2. i486™ Microprocessor I/O Instructions Clock Count Summary

INSTRUCTION	FORMAT	Real Mode	Protected Mode (CPL ≤ IOPL)	Protected Mode (CPL > IOPL)	Virtual 86 Mode	Notes
I/O INSTRUCTIONS						
IN = Input from:						
Fixed Port	1110010w port number	14	9	29	27	
Variable Port	1110110w	14	8	28	27	
OUT = Output to:						
Fixed Port	1110011w port number	16	11	31	29	
Variable Port	1110111w	16	10	30	29	
INS = Input Byte/Word from DX Port	0110110w	17	10	32	30	
OUTS = Output Byte/Word to DX Port	0110111w	17	10	32	30	1
REP INS = Input String	11110010 0110110w	16+8c	10+8c	30+8c	29+8c	2
REP OUTS = Output String	11110010 0110111w	17+5c	11+5c	31+5c	30+5c	3

NOTES:

1. Two clock cache miss penalty in all cases.
2. c = count in CX or ECX.
3. Cache miss penalty in all modes: Add 2 clocks for every 16 bytes. Entire penalty on second operation.

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit		Penalty If Cache Miss	Concurrent Execution		Notes					
		Avg (Lower Range ... Upper Range)			Avg (Lower Range ... Upper Range)							
DATA TRANSFER												
FLD = Real Load to ST(0)												
32-bit memory	<table border="1"><tr><td>11011</td><td>001</td><td>mod 000</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	001	mod 000	r/m	s-i-b/disp.	3		2			
11011	001	mod 000	r/m	s-i-b/disp.								
64-bit memory	<table border="1"><tr><td>11011</td><td>101</td><td>mod 000</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	101	mod 000	r/m	s-i-b/disp.	3		3			
11011	101	mod 000	r/m	s-i-b/disp.								
80-bit memory	<table border="1"><tr><td>11011</td><td>011</td><td>mod 101</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	011	mod 101	r/m	s-i-b/disp.	6		4			
11011	011	mod 101	r/m	s-i-b/disp.								
ST(i)	<table border="1"><tr><td>11011</td><td>001</td><td>11000</td><td>ST(i)</td><td></td></tr></table>	11011	001	11000	ST(i)		4					
11011	001	11000	ST(i)									
FILD = Integer Load to ST(0)												
16-bit memory	<table border="1"><tr><td>11011</td><td>111</td><td>mod 000</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	111	mod 000	r/m	s-i-b/disp.	14.5(13–16)		2		4	
11011	111	mod 000	r/m	s-i-b/disp.								
32-bit memory	<table border="1"><tr><td>11011</td><td>011</td><td>mod 000</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	011	mod 000	r/m	s-i-b/disp.	11.5(9–12)		2		4(2–4)	
11011	011	mod 000	r/m	s-i-b/disp.								
64-bit memory	<table border="1"><tr><td>11011</td><td>111</td><td>mod 101</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	111	mod 101	r/m	s-i-b/disp.	16.8(10–18)		3		7.8(2–8)	
11011	111	mod 101	r/m	s-i-b/disp.								
FBLD = BCD Load to ST(0)												
	<table border="1"><tr><td>11011</td><td>111</td><td>mod 100</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	111	mod 100	r/m	s-i-b/disp.	75(70–103)		4		7.7(2–8)	
11011	111	mod 100	r/m	s-i-b/disp.								
FST = Store Real from ST(0)												
32-bit memory	<table border="1"><tr><td>11011</td><td>001</td><td>mod 010</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	001	mod 010	r/m	s-i-b/disp.	7					1
11011	001	mod 010	r/m	s-i-b/disp.								
64-bit memory	<table border="1"><tr><td>11011</td><td>101</td><td>mod 010</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	101	mod 010	r/m	s-i-b/disp.	8					2
11011	101	mod 010	r/m	s-i-b/disp.								
ST(i)	<table border="1"><tr><td>11011</td><td>101</td><td>11010</td><td>ST(i)</td><td></td></tr></table>	11011	101	11010	ST(i)		3					
11011	101	11010	ST(i)									
FSTP = Store Real from ST(0) and Pop												
32-bit memory	<table border="1"><tr><td>11011</td><td>011</td><td>mod 011</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	011	mod 011	r/m	s-i-b/disp.	7					1
11011	011	mod 011	r/m	s-i-b/disp.								
64-bit memory	<table border="1"><tr><td>11011</td><td>101</td><td>mod 011</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	101	mod 011	r/m	s-i-b/disp.	8					2
11011	101	mod 011	r/m	s-i-b/disp.								
80-bit memory	<table border="1"><tr><td>11011</td><td>011</td><td>mod 111</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	011	mod 111	r/m	s-i-b/disp.	6					
11011	011	mod 111	r/m	s-i-b/disp.								
ST(i)	<table border="1"><tr><td>11011</td><td>101</td><td>11001</td><td>ST(i)</td><td></td></tr></table>	11011	101	11001	ST(i)		3					
11011	101	11001	ST(i)									
FIST = Store Integer from ST(0)												
16-bit memory	<table border="1"><tr><td>11011</td><td>111</td><td>mod 010</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	111	mod 010	r/m	s-i-b/disp.	33.4(29–34)					
11011	111	mod 010	r/m	s-i-b/disp.								
32-bit memory	<table border="1"><tr><td>11011</td><td>011</td><td>mod 010</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	011	mod 010	r/m	s-i-b/disp.	32.4(28–34)					
11011	011	mod 010	r/m	s-i-b/disp.								
FISTP = Store Integer from ST(0) and Pop												
16-bit memory	<table border="1"><tr><td>11011</td><td>111</td><td>mod 011</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	111	mod 011	r/m	s-i-b/disp.	33.4(29–34)					
11011	111	mod 011	r/m	s-i-b/disp.								
32-bit memory	<table border="1"><tr><td>11011</td><td>011</td><td>mod 011</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	011	mod 011	r/m	s-i-b/disp.	33.4(29–34)					
11011	011	mod 011	r/m	s-i-b/disp.								
64-bit memory	<table border="1"><tr><td>11011</td><td>111</td><td>mod 111</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	111	mod 111	r/m	s-i-b/disp.	33.4(29–34)					
11011	111	mod 111	r/m	s-i-b/disp.								
FBSTP = Store BCD from ST(0) and Pop												
	<table border="1"><tr><td>11011</td><td>111</td><td>mod 110</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	111	mod 110	r/m	s-i-b/disp.	175(172–176)					
11011	111	mod 110	r/m	s-i-b/disp.								
FXCH = Exchange ST(0) and ST(i)												
	<table border="1"><tr><td>11011</td><td>001</td><td>11001</td><td>ST(i)</td><td></td></tr></table>	11011	001	11001	ST(i)		4					
11011	001	11001	ST(i)									
COMPARISON INSTRUCTIONS												
FCOM = Compare ST(0) with Real												
32-bit memory	<table border="1"><tr><td>11011</td><td>000</td><td>mod 010</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	000	mod 010	r/m	s-i-b/disp.	4		2		1	
11011	000	mod 010	r/m	s-i-b/disp.								
64-bit memory	<table border="1"><tr><td>11011</td><td>100</td><td>mod 010</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	100	mod 010	r/m	s-i-b/disp.	4		3		1	
11011	100	mod 010	r/m	s-i-b/disp.								
ST(i)	<table border="1"><tr><td>11011</td><td>000</td><td>11010</td><td>ST(i)</td><td></td></tr></table>	11011	000	11010	ST(i)		4				1	
11011	000	11010	ST(i)									
FCOMP = Compare ST(0) with Real and Pop												
32-bit memory	<table border="1"><tr><td>11011</td><td>000</td><td>mod 011</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	000	mod 011	r/m	s-i-b/disp.	4		2		1	
11011	000	mod 011	r/m	s-i-b/disp.								
64-bit memory	<table border="1"><tr><td>11011</td><td>100</td><td>mod 011</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	100	mod 011	r/m	s-i-b/disp.	4		3		1	
11011	100	mod 011	r/m	s-i-b/disp.								
ST(i)	<table border="1"><tr><td>11011</td><td>000</td><td>11011</td><td>ST(i)</td><td></td></tr></table>	11011	000	11011	ST(i)		4				1	
11011	000	11011	ST(i)									

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty If Cache Miss	Concurrent Execution	Notes
		Avg (Lower Range ... Upper Range)		Avg (Lower Range ... Upper Range)	
COMPARISON INSTRUCTIONS (Continued)					
FCOMPP = Compare ST(0) with ST(1) and Pop Twice	11011 110 1101 1001	5		1	
FICOM = Compare ST(0) with Integer					
16-bit memory	11011 110 mod 010 r/m s-i-b/disp.	18(16-20)	2	1	
32-bit memory	11011 010 mod 010 r/m s-i-b/disp.	16.5(15-17)	2	1	
FICOMP = Compare ST(0) with Integer					
16-bit memory	11011 110 mod 011 r/m s-i-b/disp.	18(16-20)	2	1	
32-bit memory	11011 010 mod 011 r/m s-i-b/disp.	16.5(15-17)	2	1	
FTST = Compare ST(0) with 0.0	11011 001 1110 0100	4		1	
FUCOM = Unordered compare ST(0) with ST(i)	11011 101 11100 ST(i)	4		1	
FUCOMP = Unordered compare ST(0) with ST(i) and Pop	11011 101 11101 ST(i)	4		1	
FUCOMPP = Unordered compare ST(0) with ST(i) and Pop Twice	11011 101 11101 1001	5		1	
FXAM = Examine ST(0)	11011 001 1110 0101	8			
CONSTANTS					
FLDZ = Load +0.0 into ST(0)	11011 001 1110 1110	4			
FLD1 = Load +1.0 into ST(0)	11011 001 1110 1000	4			
FLDPI = Load π into ST(0)	11011 001 1110 1011	8		2	
FLDL2T = Load $\log_2(10)$ into ST(0)	11011 001 1110 1001	8		2	
FLDL2E = Load $\log_2(e)$ into ST(0)	11011 001 1110 1010	8		2	
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	11011 001 1110 1100	8		2	
FLDLN2 = Load $\log_e(2)$ into ST(0)	11011 001 1110 1101	8		2	
ARITHMETIC					
FADD = Add Real with ST(0)					
ST(0) \leftarrow ST(0) + 32-bit memory	11011 000 mod 000 r/m s-i-b/disp.	10(8-20)	2	7(5-17)	
ST(0) \leftarrow ST(0) + 64-bit memory	11011 100 mod 000 r/m s-i-b/disp.	10(8-20)	3	7(5-17)	
ST(d) \leftarrow ST(0) + ST(i)	11011 d00 11000 ST(i)	10(8-20)		7(5-17)	
FADDP = Add real with ST(0) and Pop (ST(i) \leftarrow ST(0) + ST(i))	11011 110 111000 ST(i)	10(8-20)		7(5-17)	
FSUB = Subtract real from ST(0)					
ST(0) \leftarrow ST(0) - 32-bit memory	11011 000 mod 100 r/m s-i-b/disp.	10(8-20)	2	7(5-17)	
ST(0) \leftarrow ST(0) - 64-bit memory	11011 100 mod 100 r/m s-i-b/disp.	10(8-20)	3	7(5-17)	
ST(d) \leftarrow ST(0) - ST(i)	11011 d00 11101 ST(i)	10(8-20)		7(5-17)	
FSUBP = Subtract real from ST(0) and Pop (ST(i) \leftarrow ST(0) - ST(i))	11011 110 11101 ST(i)	10(8-20)		7(5-17)	

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty If Cache Miss	Concurrent Execution	Notes					
		Avg (Lower Range ... Upper Range)		Avg (Lower Range ... Upper Range)						
ARITHMETIC (Continued)										
FSUBR = Subtract real reversed (Subtract ST(0) from real)										
ST(0) ← 32-bit memory – ST(0)	<table border="1"><tr><td>11011</td><td>000</td><td>mod 101</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	000	mod 101	r/m	s-i-b/disp.	10(8–20)	2	7(5–17)	
11011	000	mod 101	r/m	s-i-b/disp.						
ST(0) ← 64-bit memory – ST(0)	<table border="1"><tr><td>11011</td><td>100</td><td>mod 101</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	100	mod 101	r/m	s-i-b/disp.	10(8–20)	3	7(5–17)	
11011	100	mod 101	r/m	s-i-b/disp.						
ST(d) ← ST(i) – ST(0)	<table border="1"><tr><td>11011</td><td>d00</td><td>11100</td><td>ST(i)</td><td></td></tr></table>	11011	d00	11100	ST(i)		10(8–20)		7(5–17)	
11011	d00	11100	ST(i)							
FSUBRP = Subtract real reversed and Pop (ST(i) ← ST(i) – ST(0))	<table border="1"><tr><td>11011</td><td>110</td><td>11100</td><td>ST(i)</td><td></td></tr></table>	11011	110	11100	ST(i)		10(8–20)		7(5–17)	
11011	110	11100	ST(i)							
FMUL = Multiply real with ST(0)										
ST(0) ← ST(0) × 32-bit memory	<table border="1"><tr><td>11011</td><td>000</td><td>mod 001</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	000	mod 001	r/m	s-i-b/disp.	11	2	8	
11011	000	mod 001	r/m	s-i-b/disp.						
ST(0) ← ST(0) × 64-bit memory	<table border="1"><tr><td>11011</td><td>100</td><td>mod 001</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	100	mod 001	r/m	s-i-b/disp.	14	3	11	
11011	100	mod 001	r/m	s-i-b/disp.						
ST(d) ← ST(0) × ST(i)	<table border="1"><tr><td>11011</td><td>d00</td><td>11001</td><td>ST(i)</td><td></td></tr></table>	11011	d00	11001	ST(i)		16		13	
11011	d00	11001	ST(i)							
FMULP = Multiply ST(0) with ST(i) and Pop (ST(i) ← ST(0) × ST(i))	<table border="1"><tr><td>11011</td><td>110</td><td>11001</td><td>ST(i)</td><td></td></tr></table>	11011	110	11001	ST(i)		16		13	
11011	110	11001	ST(i)							
FDIV = Divide ST(0) by Real										
ST(0) ← ST(0)/32-bit memory	<table border="1"><tr><td>11011</td><td>000</td><td>mod 110</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	000	mod 110	r/m	s-i-b/disp.	73	2	70	3
11011	000	mod 110	r/m	s-i-b/disp.						
ST(0) ← ST(0)/64-bit memory	<table border="1"><tr><td>11011</td><td>100</td><td>mod 100</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	100	mod 100	r/m	s-i-b/disp.	73	3	70	3
11011	100	mod 100	r/m	s-i-b/disp.						
ST(d) ← ST(0)/ST(i)	<table border="1"><tr><td>11011</td><td>d00</td><td>11111</td><td>ST(i)</td><td></td></tr></table>	11011	d00	11111	ST(i)		73		70	3
11011	d00	11111	ST(i)							
FDIVP = Divide ST(0) by ST(i) and Pop (ST(i) ← ST(0)/ST(i))	<table border="1"><tr><td>11011</td><td>110</td><td>11111</td><td>ST(i)</td><td></td></tr></table>	11011	110	11111	ST(i)		73		70	3
11011	110	11111	ST(i)							
FDIVR = Divide real reversed (Real/ST(0))										
ST(0) ← 32-bit memory/ST(0)	<table border="1"><tr><td>11011</td><td>000</td><td>mod 111</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	000	mod 111	r/m	s-i-b/disp.	73	2	70	3
11011	000	mod 111	r/m	s-i-b/disp.						
ST(0) ← 64-bit memory/ST(0)	<table border="1"><tr><td>11011</td><td>100</td><td>mod 111</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	100	mod 111	r/m	s-i-b/disp.	73	3	70	3
11011	100	mod 111	r/m	s-i-b/disp.						
ST(d) ← ST(i)/ST(0)	<table border="1"><tr><td>11011</td><td>d00</td><td>11110</td><td>ST(i)</td><td></td></tr></table>	11011	d00	11110	ST(i)		73		70	3
11011	d00	11110	ST(i)							
FDIVRP = Divide real reversed and Pop (ST(i) ← ST(i)/ST(0))	<table border="1"><tr><td>11011</td><td>110</td><td>11110</td><td>ST(i)</td><td></td></tr></table>	11011	110	11110	ST(i)		73		70	3
11011	110	11110	ST(i)							
FIADD = Add Integer to ST(0)										
ST(0) ← ST(0) + 16-bit memory	<table border="1"><tr><td>11011</td><td>110</td><td>mod 000</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	110	mod 000	r/m	s-i-b/disp.	24(20–35)	2	7(5–17)	
11011	110	mod 000	r/m	s-i-b/disp.						
ST(0) ← ST(0) + 32-bit memory	<table border="1"><tr><td>11011</td><td>010</td><td>mod 000</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	010	mod 000	r/m	s-i-b/disp.	22.5(19–32)	2	7(5–17)	
11011	010	mod 000	r/m	s-i-b/disp.						
FISUB = Subtract Integer from ST(0)										
ST(0) ← ST(0) – 16-bit memory	<table border="1"><tr><td>11011</td><td>110</td><td>mod 100</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	110	mod 100	r/m	s-i-b/disp.	24(20–35)	2	7(5–17)	
11011	110	mod 100	r/m	s-i-b/disp.						
ST(0) ← ST(0) – 32-bit memory	<table border="1"><tr><td>11011</td><td>010</td><td>mod 100</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	010	mod 100	r/m	s-i-b/disp.	22.5(19–32)	2	7(5–17)	
11011	010	mod 100	r/m	s-i-b/disp.						
FISUBR = Integer Subtract Reversed										
ST(0) ← 16-bit memory – ST(0)	<table border="1"><tr><td>11011</td><td>110</td><td>mod 101</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	110	mod 101	r/m	s-i-b/disp.	24(20–35)	2	7(5–17)	
11011	110	mod 101	r/m	s-i-b/disp.						
ST(0) ← 32-bit memory – ST(0)	<table border="1"><tr><td>11011</td><td>010</td><td>mod 101</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	010	mod 101	r/m	s-i-b/disp.	22.5(19–32)	2	7(5–17)	
11011	010	mod 101	r/m	s-i-b/disp.						
FIMUL = Multiply Integer with ST(0)										
ST(0) ← ST(0) × 16-bit memory	<table border="1"><tr><td>11011</td><td>110</td><td>mod 001</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	110	mod 001	r/m	s-i-b/disp.	25(23–27)	2	8	
11011	110	mod 001	r/m	s-i-b/disp.						
ST(0) ← ST(0) × 32-bit memory	<table border="1"><tr><td>11011</td><td>010</td><td>mod 001</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	010	mod 001	r/m	s-i-b/disp.	23.5(22–24)	2	8	
11011	010	mod 001	r/m	s-i-b/disp.						
FIDIV = Integer Divide										
ST(0) ← ST(0)/16-bit memory	<table border="1"><tr><td>11011</td><td>110</td><td>mod 110</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	110	mod 110	r/m	s-i-b/disp.	87(85–89)	2	70	3
11011	110	mod 110	r/m	s-i-b/disp.						
ST(0) ← ST(0)/32-bit memory	<table border="1"><tr><td>11011</td><td>010</td><td>mod 110</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	010	mod 110	r/m	s-i-b/disp.	85.5(84–86)	2	70	3
11011	010	mod 110	r/m	s-i-b/disp.						

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Concurrent Execution	Notes
		Avg (Lower Range ... Upper Range)		Avg (Lower Range ... Upper Range)	
ARITHMETIC (Continued)					
FIDIVR = Integer Divide Reversed					
ST(0) ← 16-bit memory/ST(0)	11011 110 mod 111 r/m s-i-b/disp.	87(85-89)	2	70	3
ST(0) ← 32-bit memory/ST(0)	11011 010 mod 111 r/m s-i-b/disp.	85.5(84-86)	2	70	3
FSQRT = Square Root	11011 001 1111 1010	85.5(83-87)		70	
FSCALE = Scale ST(0) by ST(1)	11011 001 1111 1101	31(30-32)		2	
FEXTRACT = Extract components of ST(0)	11011 001 1111 0100	19(16-20)		4(2-4)	
FPREM = Partial Remainder	11011 001 1111 1000	84(70-138)		2(2-8)	
FPREM1 = Partial Remainder (IEEE)	11011 001 1111 0101	94.5(72-167)		5.5(2-18)	
FRNDINT = Round ST(0) to integer	11011 001 1111 1100	29.1(21-30)		7.4(2-8)	
FABS = Absolute value of ST(0)	11011 001 1110 0001	3			
FCHS = Change sign of ST(0)	11011 001 1110 0000	6			
TRANSCENDENTAL					
FCOS = Cosine of ST(0)	11011 001 1111 1111	241(193-279)		2	6,7
FPTAN = Partial tangent of ST(0)	11011 001 1111 0010	244(200-273)		70	6,7
FPATAN = Partial arctangent	11011 001 1111 0011	289(218-303)		5(2-17)	6
FSIN = Sine of ST(0)	11011 001 1111 1110	241(193-279)		2	6,7
FSINCOS = Sine and cosine of ST(0)	11011 001 1111 1011	291(243-329)		2	6,7
F2XM1 = 2^{ST(0)} - 1	11011 001 1111 0000	242(140-279)		2	6
FYL2X = ST(1) × log₂(ST(0))	11011 001 1111 0001	311(196-329)		13	6
FYL2XP1 = ST(1) × log₂(ST(0) + 1.0)	11011 001 1111 1001	313(171-326)		13	6
PROCESSOR CONTROL					
FINIT = Initialize FPU	11011 011 1110 0011	17			4
FSTSW AX = Store status word into AX	11011 111 1110 0000	3			5
FSTSW = Store status word into memory	11011 101 mod 111 r/m s-i-b/disp.	3			5
FLDCW = Load control word	11011 001 mod 101 r/m s-i-b/disp.	4	2		
FSTCW = Store control word	11011 001 mod 111 r/m s-i-b/disp.	3			5
FCLEX = Clear exceptions	11011 011 1110 0010	7			4
FSTENV = Store environment	11011 001 mod 110 r/m s-i-b/disp.				
Real and Virtual modes 16-bit Address		67			4
Real and Virtual modes 32-bit Address		67			4
Protected mode 16-bit Address		56			4
Protected mode 32-bit Address		56			4
FLDENV = Load environment	11011 001 mod 100 r/m s-i-b/disp.				
Real and Virtual modes 16-bit Address		44	2		
Real and Virtual modes 32-bit Address		44	2		
Protected mode 16-bit Address		34	2		
Protected mode 32-bit Address		34	2		

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty If Cache Miss	Concurrent Execution	Notes						
		Avg (Lower Range ... Upper Range)		Avg (Lower Range ... Upper Range)							
PROCESSOR CONTROL (Continued)											
FSAVE = Save state	<table border="1"><tr><td>11011</td><td>101</td><td>mod</td><td>110</td><td>r/m</td><td>s-i-b/disp.</td></tr></table>	11011	101	mod	110	r/m	s-i-b/disp.				
11011	101	mod	110	r/m	s-i-b/disp.						
Real and Virtual modes 16-bit Address		154			4						
Real and Virtual modes 32-bit Address		154			4						
Protected mode 16-bit Address		143			4						
Protected mode 32-bit Address		143			4						
FRSTOR = Restore state	<table border="1"><tr><td>11011</td><td>101</td><td>mod</td><td>100</td><td>r/m</td><td>s-i-b/</td></tr></table>	11011	101	mod	100	r/m	s-i-b/				
11011	101	mod	100	r/m	s-i-b/						
Real and Virtual modes 16-bit Address		131	23								
Real and Virtual modes 32-bit Address		131	27								
Protected mode 16-bit Address		120	23								
Protected mode 32-bit Address		120	27								
FINCSTP = Increment Stack Pointer	<table border="1"><tr><td>11011</td><td>001</td><td>1111</td><td>0111</td></tr></table>	11011	001	1111	0111	3					
11011	001	1111	0111								
FDECSTP = Decrement Stack Pointer	<table border="1"><tr><td>11011</td><td>001</td><td>1111</td><td>0110</td></tr></table>	11011	001	1111	0110	3					
11011	001	1111	0110								
FFREE = Free ST(i)	<table border="1"><tr><td>11011</td><td>101</td><td>11000</td><td>ST(i)</td></tr></table>	11011	101	11000	ST(i)	3					
11011	101	11000	ST(i)								
FNOP = No operations	<table border="1"><tr><td>11011</td><td>001</td><td>1101</td><td>0000</td></tr></table>	11011	001	1101	0000	3					
11011	001	1101	0000								
WAIT = Wait until FPU ready (Minimum/Maximum)	<table border="1"><tr><td>10011011</td></tr></table>	10011011	1/3								
10011011											

NOTES:

1. If operand is 0 clock counts = 27.
2. If operand is 0 clock counts = 28.
3. If CW.PC indicates 24 bit precision then subtract 38 clocks.
If CW.PC indicates 53 bit precision then subtract 11 clocks.
4. If there is a numeric error pending from a previous instruction add 17 clocks.
5. If there is a numeric error pending from a previous instruction add 18 clocks.
6. The INT pin is polled several times while this instruction is executing to assure short interrupt latency.
7. If ABS(operand) is greater than $\pi/4$ then add n clocks. Where $n = (\text{operand}/(\pi/4))$.

10.2 Instruction Encoding

10.2.1 OVERVIEW

All instruction encodings are subsets of the general instruction format shown in Figure 10.1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the “mod r/m” byte and “scaled index” byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

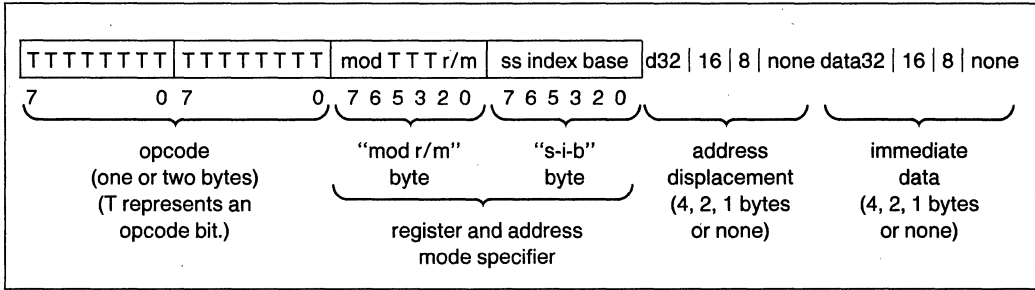
Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain

encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 10.1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 10.4 is a complete list of all fields appearing in the 486 Microprocessor instruction set. Further ahead, following Table 10.4, are detailed tables for each field.


Figure 10.1. General Instruction Format
Table 10.4. Fields within i486™ Microprocessor Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 Bits)	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field Must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
ttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

NOTE:

Tables 10.1–10.3 show encoding of individual instructions.

10.2.2 32-BIT EXTENSIONS OF THE INSTRUCTION SET

With the 486 Microprocessor, the 8086/80186/80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 bits or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bits or 16 bits) for both operands and effective addresses when executing that code segment. In the Real Address Mode or Virtual 8086 Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the 486

Microprocessor when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the opcode bytes. The presence of the Operand Size Prefix and the Effective Address Prefix will toggle the operand size or the effective address size, respectively, to the value "opposite" from the Default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. As another example, if the default effective address size is 16 bits, presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.

These 32-bit extensions are available in all 486 Microprocessor modes, including the Real Address Mode or the Virtual 8086 Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

10.2.3 ENCODING OF INTEGER INSTRUCTION FIELDS

Within the instruction are several fields indicating register selection, addressing mode and so on. The exact encodings of these fields are defined immediately ahead.

10.2.3.1 Encoding of Operand Length (w) Field

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size During 16-Bit Data Operations	Operand Size During 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

10.2.3.2 Encoding of the General Register (reg) Field

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the “mod r/m” byte, or as the r/m field of the “mod r/m” byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-Bit Data Operations	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations:		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-Bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

10.2.3.3 Encoding of the Segment Register (sreg) Field

The sreg field in certain instructions is a 2-bit field allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the 486 Microprocessor FS and GS segment registers to be specified.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

10.2.3.4 Encoding of Address Mode

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the “mod r/m” byte, and a second byte of addressing information, the “s-i-b” (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the “mod r/m” byte has $r/m = 100$ and $mod = 00, 01$ or 10 . When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the “mod r/m” byte, also contains three bits (shown as TTT in Figure 10.1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the “mod r/m” byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the “mod r/m” byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following three pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Encoding of 16-bit Address Mode with “mod r/m” Byte

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	SS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by r/m During 16-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode with “mod r/m” byte (no “s-i-b” byte present):

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by reg or r/m during 16-Bit Data Operations:

mod r/m	Function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by reg or r/m during 32-Bit Data Operations:

mod r/m	Function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode (“mod r/m” byte and “s-i-b” byte present):

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

****IMPORTANT NOTE:**

When index field is 100, indicating “no index register,” then ss field MUST equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

NOTE:

Mod field in “mod r/m” byte; ss, index, base fields in “s-i-b” byte.

10.2.3.5 Encoding of Operation Direction (d) Field

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory <- - Register "reg" Field Indicates Source Operand; "mod r/m" or "mod ss index base" Indicates Destination Operand
1	Register <- - Register/Memory "reg" Field Indicates Destination Operand; "mod r/m" or "mod ss index base" Indicates Source Operand

10.2.3.6 Encoding of Sign-Extend (s) Field

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16 32
0	None	None
1	Sign-Extend Data8 to Fill 16-Bit or 32-Bit Destination	None

10.2.3.7 Encoding of Conditional Test (ttn) Field

For the conditional instructions (conditional jumps and set on condition), ttn is encoded with n indicating to use the condition (n=0) or its negation (n=1), and ttt giving the condition to test.

Mnemonic	Condition	ttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

10.2.3.8 Encoding of Control or Debug or Test Register (eee) Field

For the loading and storing of the Control, Debug and Test registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3
Do not use any other encoding	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

When Interpreted as Test Register Field

eee Code	Reg Name
011	TR3
100	TR4
101	TR5
110	TR6
111	TR7
Do not use any other encoding	

		Instruction								Optional Fields	
		First Byte				Second Byte					
1	11011	OPA		1	mod		1	OPB	r/m	s-i-b	disp
2	11011	MF			OPA	mod		OPB		r/m	s-i-b disp
3	11011	d	P	OPA	1	1	OPB		ST(i)		
4	11011	0	0	1	1	1	1	OP			
5	11011	0	1	1	1	1	1	OP			
		15-11	10	9	8	7	6	5	4	3	2 1 0

10.2.4 ENCODING OF FLOATING POINT INSTRUCTION FIELDS

Instructions for the FPU assume one of the five forms shown in the following table. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B.

OP = Instruction opcode, possible split into two fields OPA and OPB

MF = Memory Format
 00—32-bit real
 01—32-bit integer
 10—64-bit real
 11—16-bit integer

P = Pop
 0—Do not pop stack
 1—Pop stack after operation

d = Destination
 0—Destination is ST(0)
 1—Destination is ST(i)

R XOR d = 0—Destination (op) Source
 R XOR d = 1—Source (op) Destination

ST(i) = Register stack element *i*
 000 = Stack top
 001 = Second stack element
 •
 •
 •
 111 = Eighth stack element

mod (Mode field) and r/m (Register/Memory specifier) have the same interpretation as the corresponding fields of the integer instructions.

s-i-b (Scale Index Base) byte and disp (displacement) are optionally present in instructions that have mod and r/m fields. Their presence depends on the values of mod and r/m, as for integer instructions.

11.0 DIFFERENCES BETWEEN THE i486™ MICROPROCESSOR AND THE 386™ MICROPROCESSOR PLUS THE 387™ MATH COPROCESSOR EXTENSION

The differences between the 486 microprocessor and the 386 microprocessor are due to performance enhancements. The differences between the microprocessors are listed below.

1. Instruction clock counts have been reduced to achieve higher performance. See Section 10.
2. The 486 microprocessor bus is significantly faster than the 386 microprocessor bus. Differences include a 1X clock, parity support, burst cycles, cacheable cycles, cache invalidate cycles and 8-bit bus support. The Hardware Interface and Bus Operation Sections (Sections 6 and 7) of the data sheet should be carefully read to understand the 486 microprocessor bus functionality.
3. To support the on-chip cache new bits have been added to control register 0 (CD and NW) (Section 2.1.2.1), new pins have been added to the bus (Section 6) and new bus cycle types have been added (Section 7). The on-chip cache needs to be enabled after reset by clearing the CD and NW bit in CR0.
4. The complete 387 math coprocessor instruction set and register set have been added. No I/O cycles are performed during Floating Point instructions. The instruction and data pointers are set to 0 after FINIT/FSAVE. Interrupt 9 can no longer occur, interrupt 13 occurs instead.
5. The 486 microprocessor supports new floating point error reporting modes to guarantee DOS compatibility. These new modes required a new bit in control register 0 (NE) (Section 2.1.2.1) and new pins (FERR# and IGNNE#) (Section 6.2.13 and 7.2.14).

6. Six new instructions have been added:
 - Byte Swap (BSWAP)
 - Exchange-and-Add (XADD)
 - Compare and Exchange (CMPXCHG)
 - Invalidate Data Cache (INVD)
 - Write-back and Invalidate Data Cache (WBINVD)
 - Invalidate TLB Entry (INVLPG)
7. There are two new bits defined in control register 3, the page table entries and page directory entries (PCD and PWT) (Section 4.5.2.5).
8. A new page protection feature has been added. This feature required a new bit in control register 0 (WP) (Section 2.1.2.1 and 4.5.3).
9. A new Alignment Check feature has been added. This feature required a new bit in the flags register (AC) (Section 2.1.1.3) and a new bit in control register 0 (AM) (Section 2.1.2.1).
10. The replacement algorithm for the translation lookaside buffer has been changed to a pseudo least recently used algorithm like that used by the on-chip cache. See Section 5.5 for a description of the algorithm.
11. Three new testability registers, TR3, TR4 and TR5, have been added for testing the on-chip cache. TLB testability has been enhanced. See Section 8.
12. The prefetch queue has been increased from 16 bytes to 32 bytes. A jump always needs to execute after modifying code to guarantee correct execution of the new instruction.
13. After reset, the ID in the upper byte of the DX register is 04. The contents of the base registers including the floating point registers may be different after reset.

12.0 ELECTRICAL DATA

The following sections describe recommended electrical connections for the 486 microprocessor, and its electrical specifications.

12.1 Power and Grounding

12.1.1 POWER CONNECTIONS

The 486 microprocessor is implemented in CHMOS IV technology and has modest power requirements.

However, its high clock frequency output buffers can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 24 V_{CC} and 28 V_{SS} pins feed the 486 microprocessor.

Power and ground connections must be made to all external V_{CC} and GND pins of the 486 microprocessor. On the circuit board, all V_{CC} pins must be connected on a V_{CC} plane. All V_{SS} pins must be likewise connected on a GND plane.

12.1.2 POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitance should be placed near the 486 microprocessor. The 486 microprocessor driving its 32-bit parallel address and data buses at high frequencies can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the 486 microprocessor and decoupling capacitors as much as possible. Capacitors specifically for PGA packages are also commercially available.

12.1.3 OTHER CONNECTION RECOMMENDATIONS

N.C. pins should always remain unconnected.

For reliable operation, always connect unused inputs to an appropriate signal level. Active LOW inputs should be connected to V_{CC} through a pullup resistor. Pullups in the range of 20 K Ω are recommended. Active HIGH inputs should be connected to GND.

12.2 Maximum Ratings

Table 12.1 is a stress rating only, and functional operation at the maximums is not guaranteed. Function operating conditions are given in 12.3 D.C. Specifications and 12.4 A.C. Specifications.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the 486 microprocessor contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

Table 12.1. Absolute Maximum Ratings

Case Temperature under Bias . . . -65°C to $+110^{\circ}\text{C}$
 Storage Temperature -65°C to $+150^{\circ}\text{C}$
 Voltage on Any Pin with
 Respect to Ground -0.5 to $V_{\text{CC}} + 0.5\text{V}$
 Supply Voltage with
 Respect to V_{SS} -0.5V to $+6.5\text{V}$

12.3 D.C. Specifications

Functional Operating Range: $V_{\text{CC}} = 5\text{V} \pm 5\%$; $T_{\text{CASE}} = 0^{\circ}\text{C}$ to $+85^{\circ}\text{C}$

Table 12.2. DC Parametric Values

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	-0.3	$+0.8$	V	
V_{IH}	Input High Voltage	2.0	$V_{\text{CC}} + 0.3$	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current (25 MHz)		700	mA	(Note 3)
	Power Supply Current (33 MHz)		900		
I_{LI}	Input Leakage Current		± 15	μA	(Note 4)
I_{IH}	Input Leakage Current		200	μA	(Note 5)
I_{IL}	Input Leakage Current		-400	μA	(Note 6)
I_{LO}	Output Leakage Current		± 15	μA	
C_{IN}	Input Capacitance		20	pF	$F_{\text{C}} = 1\text{ MHz}$ (Note 7)
C_{O}	I/O or Output Capacitance		20	pF	$F_{\text{C}} = 1\text{ MHz}$ (Note 7)
C_{CLK}	CLK Capacitance		20	pF	$F_{\text{C}} = 1\text{ MHz}$ (Note 7)

NOTES:

- This parameter is measured at:
 Address, Data, BEn 4.0 mA
 Definition, Control 5.0 mA
- This parameter is measured at:
 Address, Data, BEn 1.0 mA
 Definition, Control 0.9 mA
- Typical supply current:
 550 mA @ 25 MHz
 700 mA @ 33 MHz
- This parameter is for inputs without pullups or pulldowns and $0 \leq V_{\text{IN}} \leq V_{\text{CC}}$.
- This parameter is for inputs with pulldowns and $V_{\text{IH}} = 2.4\text{V}$.
- This parameter is for inputs with pullups and $V_{\text{IL}} = 0.45\text{V}$.
- Not 100% tested.

12.4 A.C. Specifications

The A.C. specifications, given in Table 12.3, consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the CLK signal.

A.C. specifications measurement is defined by Figures 12.1–12.3. Inputs must be driven to the voltage levels indicated by Figure 12.3 when A.C. specifica-

tions are measured. 486 microprocessor output delays are specified with minimum and maximum limits, measured as shown. The minimum 486 microprocessor delay times are hold times provided to external circuitry. 486 microprocessor input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 486 microprocessor operation.

Table 12.3. 25 MHz i486 Microprocessor A.C. Characteristics
 $V_{CC} = 5V \pm 5\%$; $T_{case} = 0^{\circ}C$ to $+85^{\circ}C$; $C_l = 50$ pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	25	MHz		1X Clock Driven to 486
t_1	CLK Period	40	125	ns	12.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	14		ns	12.1	at 2V
t_3	CLK Low Time	14		ns	12.1	at 0.8V
t_4	CLK Fall Time		4	ns	12.1	
t_5	CLK Rise Time		4	ns	12.1	
t_6	A2-A31, PWT, PCD, BE0-3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	22	ns	12.2	
t_7	A2-A31, PWT, PCD, BE0-3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		30	ns	12.2	Note 1
t_8	PCHK# Valid Delay	3	27	ns	12.2	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	27	ns	12.2	
t_9	BLAST#, PLOCK# Float Delay		30	ns	12.2	Note 1
t_{10}	D0-D31, DP0-3 Write Data Valid Delay	3	22	ns	12.2	
t_{11}	D0-D31, DP0-3 Write Data Float Delay		30	ns	12.2	Note 1
t_{12}	EADS# Setup Time	8		ns	12.3	
t_{13}	EADS# Hold Time	3		ns	12.3	
t_{14}	KEN#, BS16#, BS8# Setup Time	8		ns	12.3	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	12.3	
t_{16}	RDY#, BRDY# Setup Time	8		ns	12.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	12.3	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	10		ns	12.3	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	12.3	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	10		ns	12.3	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	12.3	
t_{22}	D0-D31, DP0-3, A4-A31 Read Setup Time	5		ns	12.3	
t_{23}	D0-D31, DP0-3, A4-A31 Read Hold Time	3		ns	12.3	

NOTE:

1. Not 100% tested. Guaranteed by design characterization.

Table 12.3. 33 MHz i486 Microprocessor A.C. Characteristics
 $V_{CC} = 5V \pm 5\%$; $T_{case} = 0^{\circ}C$ to $+85^{\circ}C$; $C_l = 50$ pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	33	MHz		1X Clock Driven to 486
t ₁	CLK Period	30	125	ns	12.1	
t _{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t ₂	CLK High Time	11		ns	12.1	at 2V
t ₃	CLK Low Time	11		ns	12.1	at 0.8V
t ₄	CLK Fall Time		3	ns	12.1	
t ₅	CLK Rise Time		3	ns	12.1	
t ₆	A2-A31, PWT, PCD, BE0-3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	16	ns	12.2	*
t ₇	A2-A31, PWT, PCD, BE0-3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		20	ns	12.2	Note 1
t ₈	PCHK# Valid Delay	3	22	ns	12.2	
t _{8a}	BLAST#, PLOCK# Valid Delay	3	20	ns	12.2	
t ₉	BLAST#, PLOCK# Float Delay		20	ns	12.2	Note 1
t ₁₀	D0-D31, DP0-3 Write Data Valid Delay	3	18	ns	12.2	
t ₁₁	D0-D31, DP0-3 Write Data Float Delay		20	ns	12.2	Note 1
t ₁₂	EADS# Setup Time	5		ns	12.3	
t ₁₃	EADS# Hold Time	3		ns	12.3	
t ₁₄	KEN#, BS16#, BS8# Setup Time	5		ns	12.3	
t ₁₅	KEN#, BS16#, BS8# Hold Time	3		ns	12.3	
t ₁₆	RDY#, BRDY# Setup Time	5		ns	12.3	
t ₁₇	RDY#, BRDY# Hold Time	3		ns	12.3	
t ₁₈	HOLD, AHOLD, Setup Time	6		ns	12.3	
t _{18a}	BOFF# Setup Time	8		ns	12.3	
t ₁₉	HOLD, AHOLD, BOFF# Hold Time	3		ns	12.3	
t ₂₀	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	5		ns	12.3	
t ₂₁	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	12.3	
t ₂₂	D0-D31, DP0-3, A4-A31 Read Setup Time	5		ns	12.3	
t ₂₃	D0-D31, DP0-3, A4-A31 Read Hold Time	3		ns	12.3	

NOTE:

1. Not 100% tested. Guaranteed by design characterization.

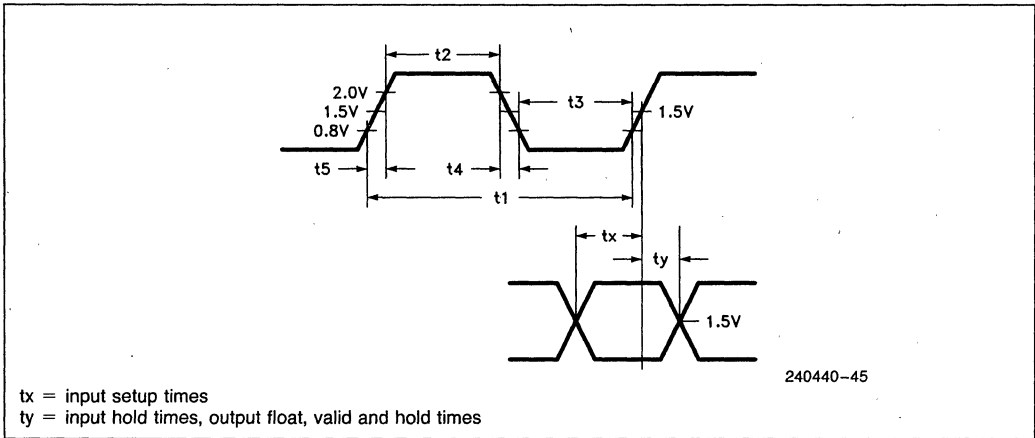


Figure 12.1. CLK Waveforms

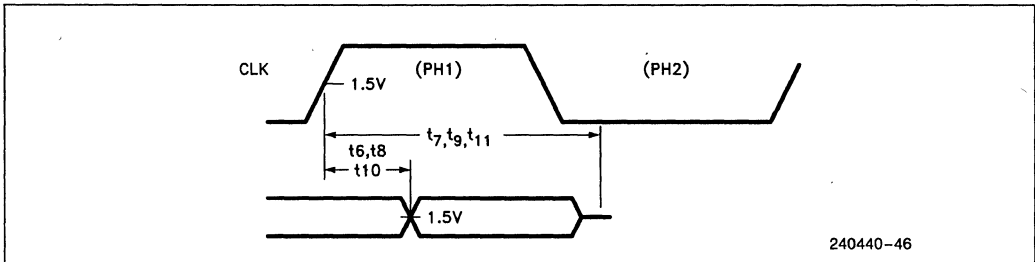


Figure 12.2. Output Waveforms

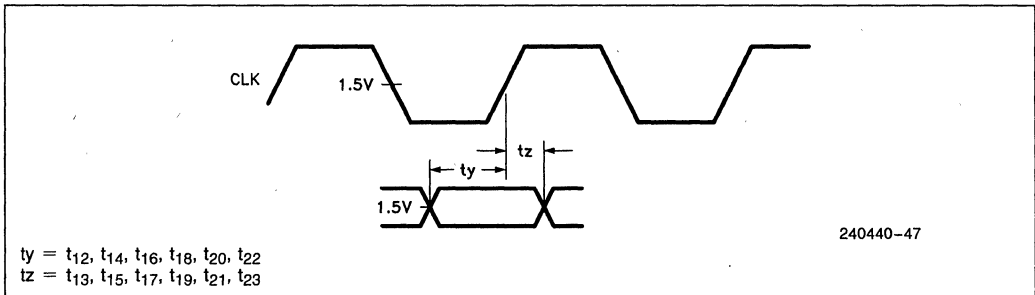
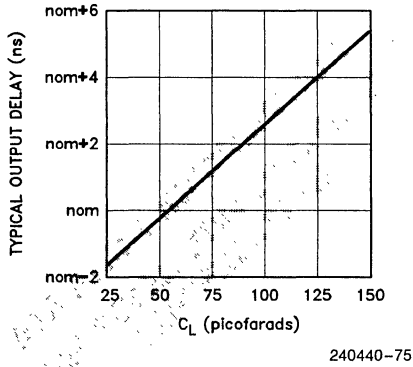


Figure 12.3. Input Waveforms

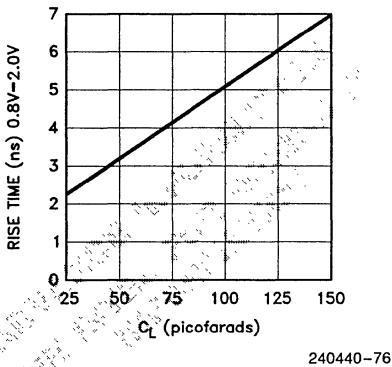
12.4.1 Typical Output Valid Delay versus Load Capacitance Under Worst Case Conditions



NOTE:

This graph will not be linear outside of the C_L range shown. non = nominal value given in A.C. Characteristics table.

12.4.2 Typical Output Rise Time versus Load Capacitance Under Worst-Case Conditions



NOTE:

This graph will not be linear outside of the C_L range shown.

12.5 Designing for ICD-486 (Advance Information)

The ICD-486 (In-Circuit Debugger) is a hardware assisted debugger for the 486 CPU. To use the ICD-

486, the 486 CPU component must be removed from its socket replaced with the ICD-486 module. Because of the high operating frequency of 486 CPU systems, there is no buffering of signals between the 486 CPU in the ICD-486 and the target system. A direct result of the non-buffered interconnect is that the ICD-486 shares the address and data bus of the target system. In order for the ICD-486 to function properly (without the Optional Isolation Board installed), the design of the target system must meet the following restrictions:

1. The bus controller must only enable data transceivers onto the data bus during valid read cycles of the 486 CPU, other local devices, or other bus masters.
2. Before another bus master drives the local processor address bus, the other bus master must gain access to the address bus through the use of HOLD-HLDA, AHOLD, or BOFF#.

In addition to the above restrictions, the ICD-486 has several electrical and mechanical characteristics that should be taken into consideration when designing the 486 CPU system.

Capacitive Loading: ICD-486 adds up to 30 pF to the CLK signal, and up to 20 pF to each of the other 486 CPU signals.

DC Loading: ICD-486 adds $\pm 15 \mu A$ loading to the CLK and data bus signals and $\pm 5 \mu A$ loading to the address and control signals.

Power Requirements: For noise immunity and CMOS latch-up protection the ICD-486 is powered by the target system through the power and ground pins of the 486 CPU socket. The circuitry on the ICD-486 draws up to 1.3A excluding the 486 CPU I_{CC} .

No Connects: Pins specified as N.C. in the 486 CPU pin description must be left unconnected. Connection of any of these pins to power, ground, or any other signal may cause the processor or the ICD-486 to malfunction.

486 CPU Location and Orientation: The ICD-486 may require lateral clearance. Figure 12.4 shows the clearance requirements of the ICD-486.

Optional Isolation Board (OIB)

Due to its unbuffered design, the ICD-486 is susceptible to errors on the target system's bus. The OIB installs between the ICD-486 and 486 CPU socket in the target system and allows the ICD-486 to function in systems with faults (i.e., shorted signals). After electrical verification the OIB may be removed. The OIB has the following electrical and mechanical characteristics:

Buffer Characteristics: The OIB buffers the address and data busses as well as the byte enables, ADS#, W/R#, M/IO#, BLAST#, and HLDA. The buffers are advanced CMOS devices and have the following DC drive specifications: $I_{OH} = -15$ mA, $I_{OL} = 64$ mA. The propagation delay of each buffer is 5 ns max driving a 50 pF load. To guarantee proper oper-

ation with the OIB, the clock period should be increased by the round trip buffer delay (10 ns) unless the target system design already has enough timing margin.

Unbuffered Signals: Signals not listed above as buffered are passed through the OIB and will have additional capacitive loading due to the connectors and circuit board of up to 10 pF.

Power Requirements: The OIB is also powered by the target system through the 486 CPU socket and requires 0.5A in addition to the ICD-486 and 486 CPU requirements.

OIB Clearance Requirements: The OIB requires an extra 0.55" of vertical clearance in the target system above the 486 CPU socket.

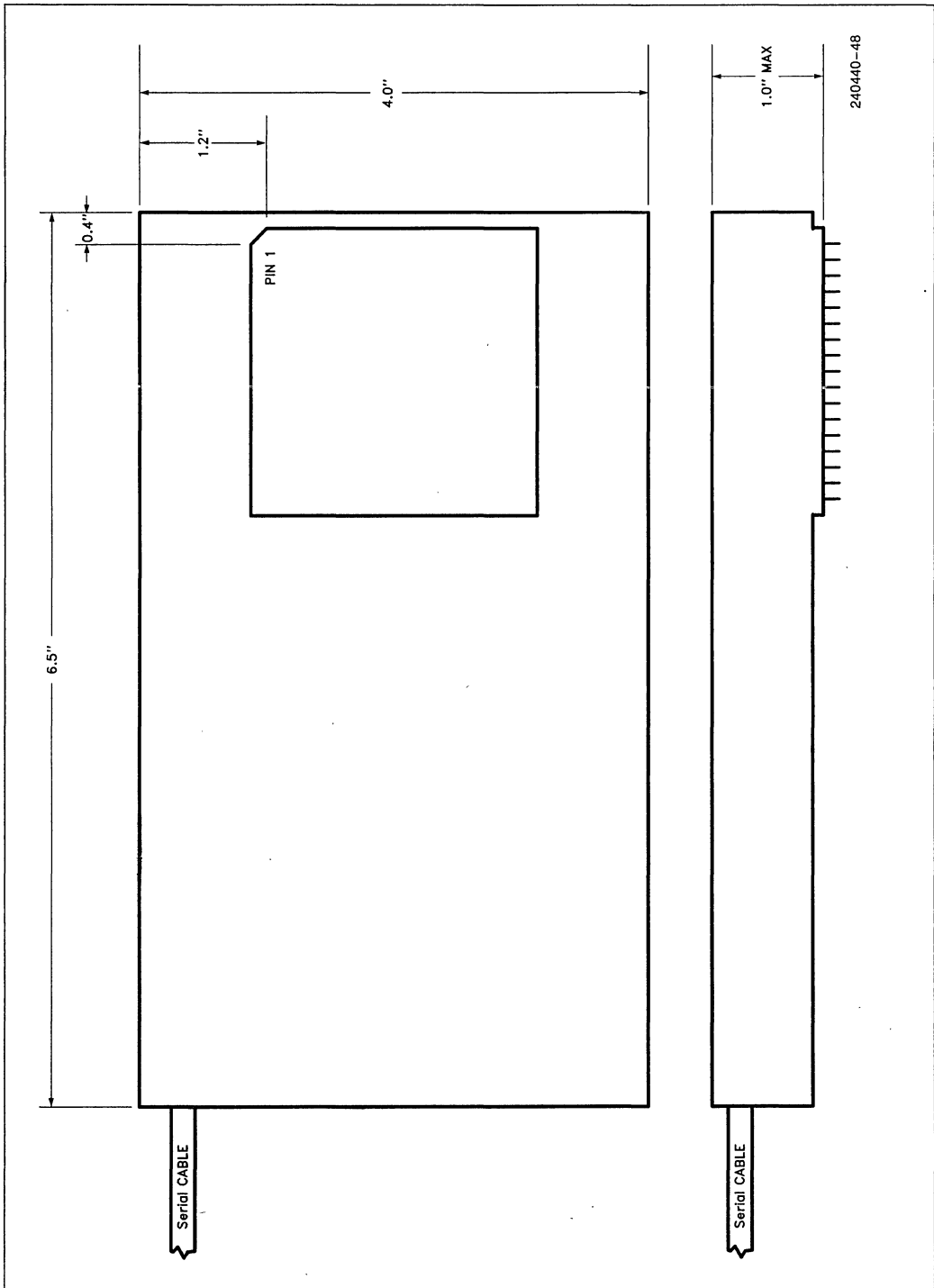


Figure 12.4a. ICD-486™ Probe Dimensions

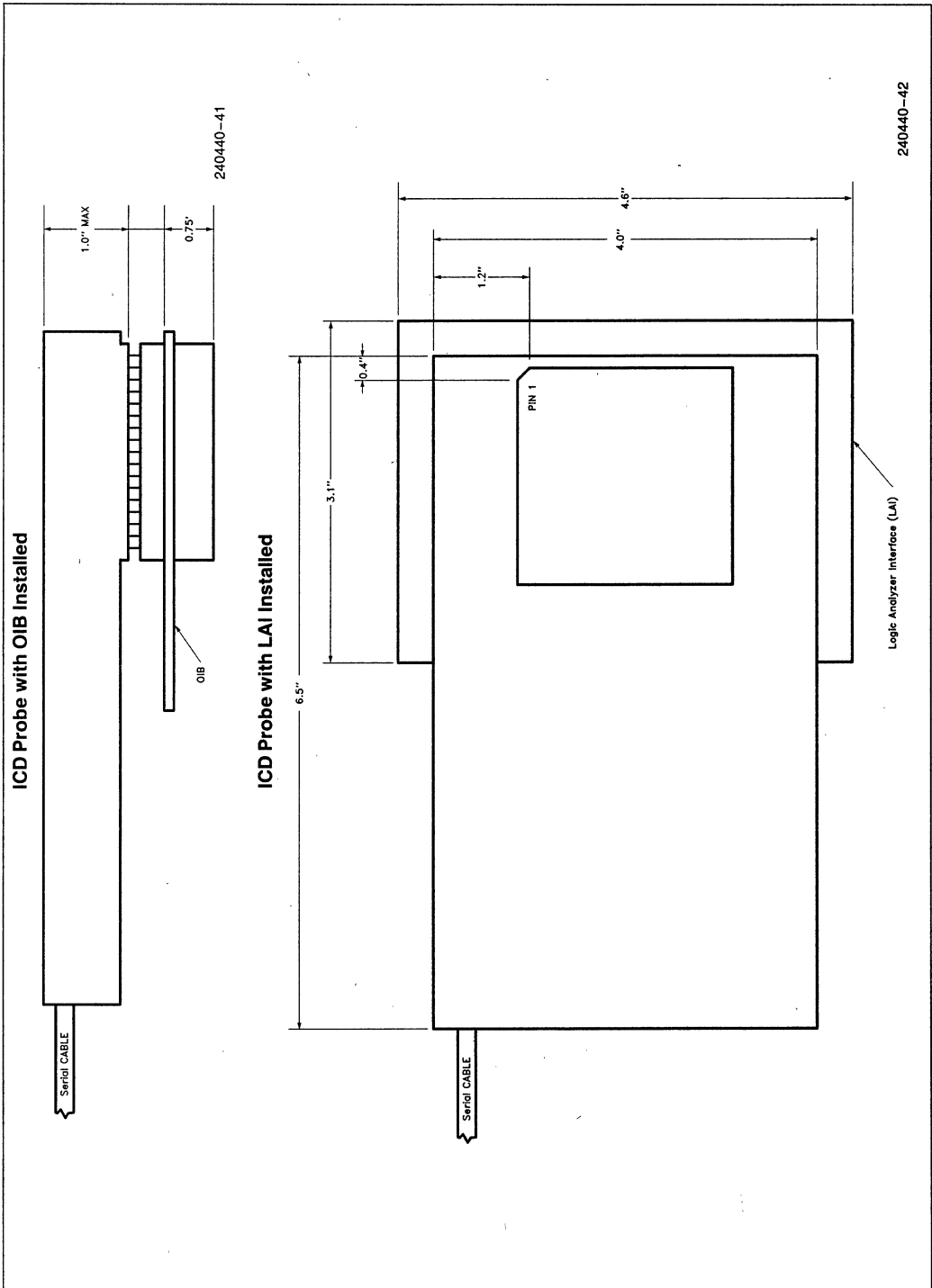
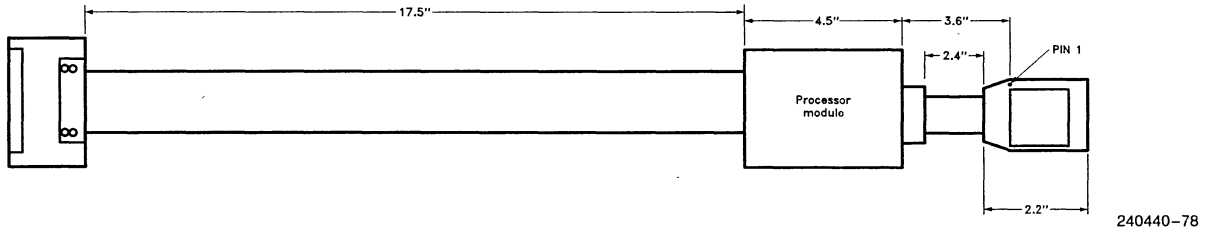


Figure 12.4b. ICD-486™ Probe Dimensions

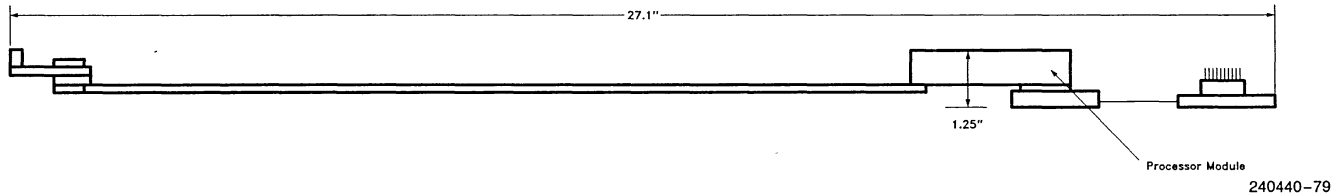
Processor Module Board Dimensions



Processor Module Assembly Dimensions Top View



Processor Module Assembly Dimensions Side View



Processor Module Assembly Dimensions Side View, OIB Installed

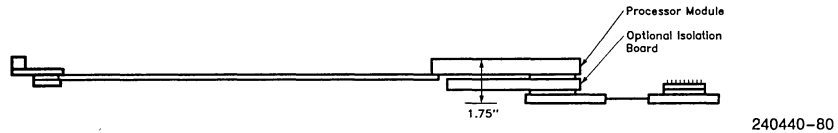
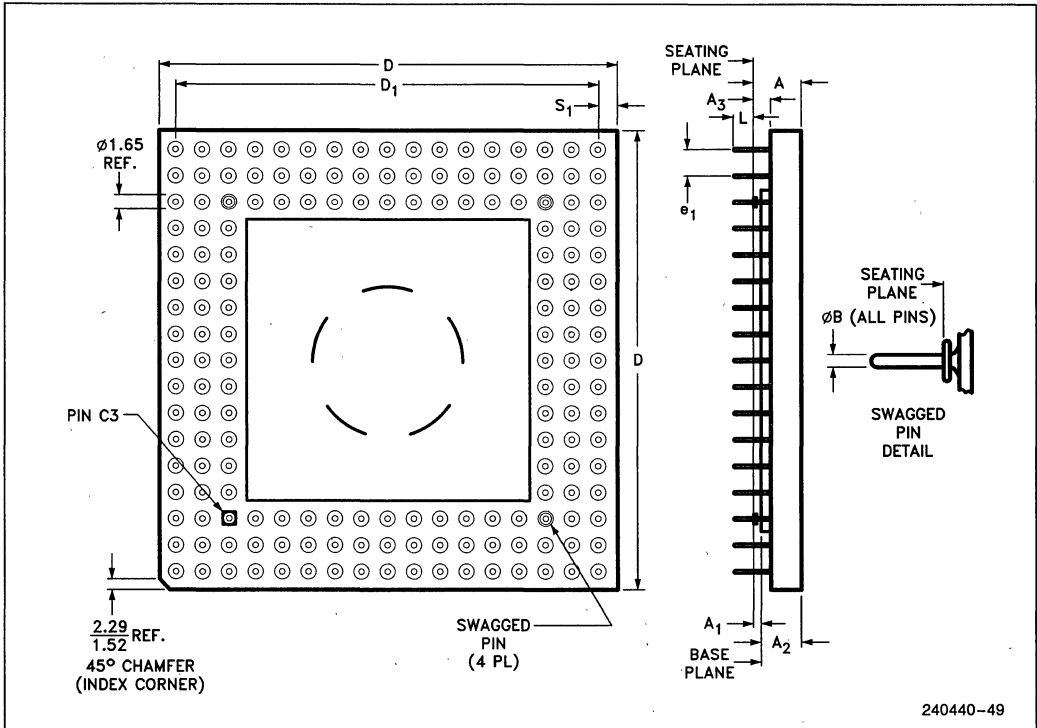


Figure 12.4c. ICD-486™ Probe Dimensions

13.0 MECHANICAL DATA



240440-49

Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		0.140	0.180	
A ₁	0.64	1.14	SOLID LID	0.025	0.045	SOLID LID
A ₂	23	0.30	SOLID LID	0.110	0.140	SOLID LID
A ₃	1.14	1.40		0.045	0.055	
B	0.43	0.51		0.017	0.020	
D	44.07	44.83		1.735	1.765	
D ₁	40.51	40.77		1.595	1.605	
e ₁	2.29	2.79		0.090	0.110	
L	2.54	3.30		0.100	0.130	
N	168			168		
S ₁	1.52	2.54		0.060	0.100	
ISSUE	IWS REV X 7/15/88					

Figure 13.1. 168 Lead Ceramic PGA Package Dimensions

Table 13.1 Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A ₁	Distance between seating plane and base plane (lid)
A ₂	Distance from base plane to highest point of body
A ₃	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D ₁	A body length dimension, outer lead center to outer lead center
e ₁	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S ₁	Other body dimension, outer lead center to edge of body

NOTES:

1. Controlling dimension: millimeter.
2. Dimension "e₁" ("e") is non-cumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B₁" and "C" are nominal.
5. Details of Pin 1 identifier are optional.

13.1 Package Thermal Specifications

The 486 microprocessor is specified for operation when T_C (the case temperature) is within the range of 0°C–85°C. T_C may be measured in any environment to determine whether the 486 microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

T_A (the ambient temperature) can be calculated from θ_{CA} (thermal resistance from case to ambient) with the following equation:

$$T_A = T_C - P * \theta_{CA}$$

Typical values for θ_{CA} at various airflows are given in Table 13.2 for the 1.75 sq. in., 168 pin, ceramic PGA.

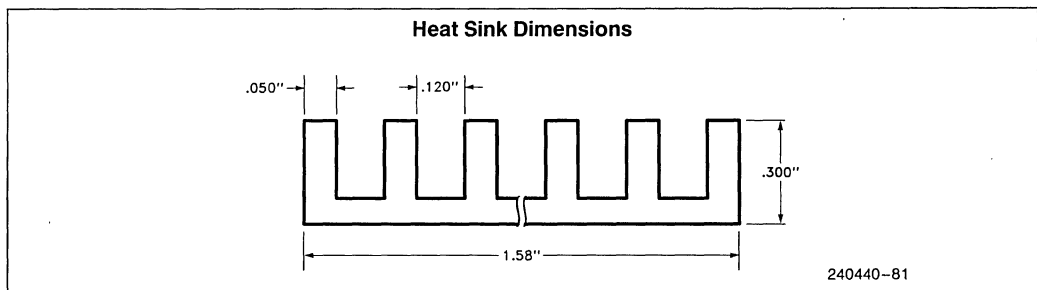
Table 13.3 shows the T_A allowable (without exceeding T_C) at various airflows and operating frequencies (f_{CLK}).

Note that T_A is greatly improved by attaching "fins" or a "heat sink" to the package. P (the maximum power consumption) is calculated by using the maximum I_{CC} at 5V as tabulated in the *DC Characteristics* of Section 12.

Table 13.2. Thermal Resistance (θ_{CA}) at Various Airflows

	In °C/Watt					
	Airflow-ft/min (m/sec)					
	0 (0)	200 (1)	400 (2)	600 (3)	800 (4)	1000 (5)
θ _{CA} with Heat Sink* (°C/W)	12	7.5	5.5	4.5	3.5	3.0
θ _{CA} without Heat Sink (°C/W)	15.5	13.0	11.0	9.5	8.5	8.0

*0.300" high unidirectional heat sink (Al alloy 6061, 50 mil fin width, 170 mil center-to-center fin spacing).



240440-81

Table 13.3. Maximum T_A at Various Airflows

	f _{CLK} (MHz)	In °C					
		Airflow-ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T _A with Heat Sink	25.0	43	59	66	69	73	75
	33.3	31	51	60	65	69	72
T _A without Heat Sink	25.0	31	40	47	52	55	57
	33.3	15	27	36	42	47	49

14.0 REVISION HISTORY

Revision -002 of the i486 CPU data sheet contains many updates and improvements to the original version. A revision summary of major changes is listed below:

- Section 2.1.2** The polarity and names of the two cache control bits in Control Register 0 (CR0) have been modified. The Cache Enable (CE) and Writes Transparent (WR) have been renamed Cache Disable (CD) and Not Write Through (NW). The value of CR0 after RESET has been changed to reflect the polarity change.
- Section 6.2.15** The discussion of A20M# has been clarified. During the falling edge of RESET, A20M# should be high, for proper operation of the CPU.
- Section 6.5** The value of CR0 after RESET has been modified.
- Section 6.5.1** Figure 6.3, "Pin State during RESET" is added. This Figure is a general reference for Reset issues. Previous Figures 8.1, 8.2, and 8.8 have been deleted, since Figure 6.3 now contains Reset information.
- Section 7.2.10** A discussion of addresses and byte enables driven during INTA cycles has been added.
- Section 10.1** Clock counts and opcodes have been clarified and corrected.
- Section 10.1** The opcode slot for CMPXCHG instruction has been moved from 0FA6/A7 to 0FB0/B1.
- Section 12.2** Table 12.1 has been enhanced. The "Case Temperature under Bias" spec was improved. The "Supply Voltage with Respect to V_{SS}" spec was added.
- Section 12.3** Maximum I_{CC} values have been improved to 700 mA at 25 MHz and 900 mA at 33 MHz.
- Section 12.3** Typical I_{CC} values have been modified to 550 mA at 25 MHz and 700 mA at 33 MHz.
- Section 12.3** C_{IN}, C_O, and C_{CLK} values have been changed to 20 pF. Testing parameters and Note 7 were added.
- Section 12.4** The A.C. Specifications have been improved. Float delays were improved at both 25 MHz and 33 MHz. Note 1 was added to the float delays. Maximum valid delays were reduced at 33 MHz.
- Section 12.5** The ICD section was enhanced.
- Section 13.1** Thermal resistance θ_{CA} values of the 168-pin ceramic package have been corrected.
- Section 13.1** Maximum ambient temperatures have been corrected to use the max I_{CC} values.



386™ DX MICROPROCESSOR

HIGH PERFORMANCE 32-BIT CHMOS MICROPROCESSOR WITH INTEGRATED MEMORY MANAGEMENT

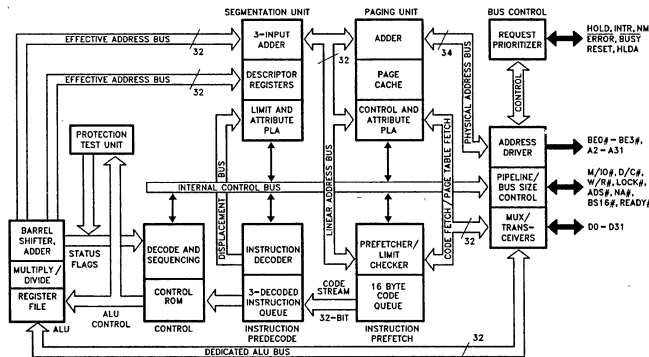
- **Flexible 32-Bit Microprocessor**
 - 8, 16, 32-Bit Data Types
 - 8 General Purpose 32-Bit Registers
- **Very Large Address Space**
 - 4 Gigabyte Physical
 - 64 Terabyte Virtual
 - 4 Gigabyte Maximum Segment Size
- **Integrated Memory Management Unit**
 - Virtual Memory Support
 - Optional On-Chip Paging
 - 4 Levels of Protection
 - Fully Compatible with 80286
- **Object Code Compatible with All 8086 Family Microprocessors**
- **Virtual 8086 Mode Allows Running of 8086 Software in a Protected and Paged System**
- **Hardware Debugging Support**
- **Optimized for System Performance**
 - Pipelined Instruction Execution
 - On-Chip Address Translation Caches
 - 20, 25 and 33 MHz Clock
 - 40, 50 and 66 Megabytes/Sec Bus Bandwidth
- **High Speed Numerics Support via 387 DX™ Coprocessor**
- **Complete System Development Support**
 - Software: C, PL/M, Assembler System Generation Tools
 - Debuggers: PSCOPE, ICETM-386
- **High Speed CHMOS III and CHMOS IV Technology**
- **132 Pin Grid Array Package**

(See Packaging Specification, Order # 231369)

The 386™ DX Microprocessor is an advanced 32-bit microprocessor designed for applications needing very high performance and optimized for multitasking operating systems. The 32-bit registers and data paths support 32-bit addresses and data types. The processor addresses up to four gigabytes of physical memory and 64 terabytes (2⁴⁶) of virtual memory. The integrated memory management and protection architecture includes address translation registers, advanced multitasking hardware and a protection mechanism to support operating systems. In addition, the 386 DX allows the simultaneous running of multiple operating systems. Instruction pipelining, on-chip address translation, and high bus bandwidth ensure short average instruction execution times and high system throughput.

The 386 DX offers new testability and debugging features. Testability features include a self-test and direct access to the page translation cache. Four new breakpoint registers provide breakpoint traps on code execution or data accesses, for powerful debugging of even ROM-based systems.

Object-code compatibility with all 8086 family members (8086, 8088, 80186, 80188, 80286) means the 386 DX offers immediate access to the world's largest microprocessor software base.



231630-49

386™ DX Pipelined 32-Bit Microarchitecture

386™ DX and 387™ DX are Trademarks of Intel Corporation.
 UNIX™ is a Trademark of AT&T Bell Labs.
 MS-DOS is a Trademark of MICROSOFT Corporation.

1. PIN ASSIGNMENT

The 386 DX pinout as viewed from the top side of the component is shown by Figure 1-1. Its pinout as viewed from the Pin side of the component is Figure 1-2.

V_{CC} and GND connections must be made to multiple V_{CC} and V_{SS} (GND) pins. Each V_{CC} and V_{SS} must be connected to the appropriate voltage level. The circuit board should include V_{CC} and GND planes for power distribution and all V_{CC} and GND pins must be connected to the appropriate plane.

NOTE:

Pins identified as "N.C." should remain completely unconnected.

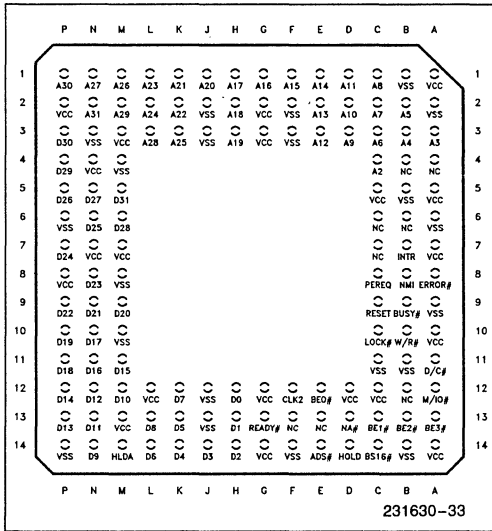


Figure 1-1. 386™ DX PGA Pinout—View from Top Side

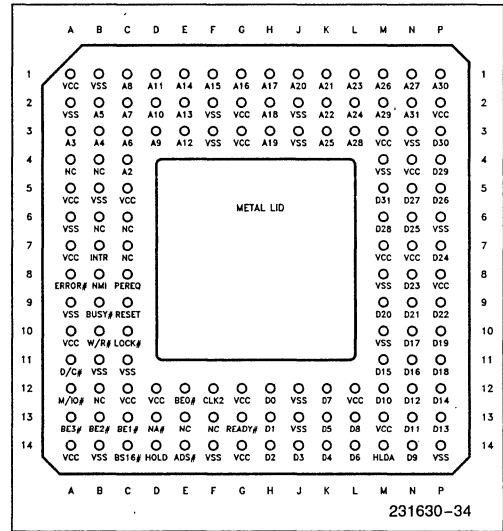


Figure 1-2. 386™ DX PGA Pinout—View from Pin Side

Table 1-1. 386™ DX PGA Pinout—Functional Grouping

Signal/Pin	Signal/Pin	Signal/Pin	Signal/Pin	Signal/Pin	Signal/Pin
A2 C4	A24 L2	D6 L14	D28 M6	V _{CC} C12	V _{SS} F2
A3 A3	A25 K3	D7 K12	D29 P4	D12	F3
A4 B3	A26 M1	D8 L13	D30 P3	G2	F14
A5 B2	A27 N1	D9 N14	D31 M5	G3	J2
A6 C3	A28 L3	D10 M12	D/C# A11	G12	J3
A7 C2	A29 M2	D11 N13	ERROR# A8	G14	J12
A8 C1	A30 P1	D12 N12	HLDA M14	L12	J13
A9 D3	A31 N2	D13 P13	HOLD D14	M3	M4
A10 D2	ADS# E14	D14 P12	INTR B7	M7	M8
A11 D1	BE0# E12	D15 M11	LOCK# C10	M13	M10
A12 E3	BE1# C13	D16 N11	M/IO# A12	N4	N3
A13 E2	BE2# B13	D17 N10	NA# D13	N7	P6
A14 E1	BE3# A13	D18 P11	NMI B8	P2	P14
A15 F1	BS16# C14	D19 P10	PEREQ C8	P8	W/R# B10
A16 G1	BUSY# B9	D20 M9	READY# G13	V _{SS} A2	N.C. A4
A17 H1	CLK2 F12	D21 N9	RESET C9	A6	B4
A18 H2	D0 H12	D22 P9	V _{CC} A1	A9	B6
A19 H3	D1 H13	D23 N8	A5	B1	B12
A20 J1	D2 H14	D24 P7	A7	B5	C6
A21 K1	D3 J14	D25 N6	A10	B11	C7
A22 K2	D4 K14	D26 P5	A14	B14	E13
A23 L1	D5 K13	D27 N5	C5	C11	F13

1.1 PIN DESCRIPTION TABLE

The following table lists a brief description of each pin on the 386 DX. The following definitions are used in these descriptions:

- # The named signal is active LOW.
- I Input signal.
- O Output signal.
- I/O Input and Output signal.
- No electrical connection.

For a more complete description refer to Section 5.2 Signal Description.

Symbol	Type	Name and Function
CLK2	I	CLK2 provides the fundamental timing for the 386 DX.
D ₃₁ -D ₀	I/O	DATA BUS inputs data during memory, I/O and interrupt acknowledge read cycles and outputs data during memory and I/O write cycles.
A ₃₁ -A ₂	O	ADDRESS BUS outputs physical memory or port I/O addresses.
BE0# - BE3#	O	BYTE ENABLES indicate which data bytes of the data bus take part in a bus cycle.
W/R#	O	WRITE/READ is a bus cycle definition pin that distinguishes write cycles from read cycles.
D/C#	O	DATA/CONTROL is a bus cycle definition pin that distinguishes data cycles, either memory or I/O, from control cycles which are: interrupt acknowledge, halt, and instruction fetching.
M/IO#	O	MEMORY I/O is a bus cycle definition pin that distinguishes memory cycles from input/output cycles.
LOCK#	O	BUS LOCK is a bus cycle definition pin that indicates that other system bus masters are denied access to the system bus while it is active.
ADS#	O	ADDRESS STATUS indicates that a valid bus cycle definition and address (W/R#, D/C#, M/IO#, BE0#, BE1#, BE2#, BE3# and A ₃₁ -A ₂) are being driven at the 386 DX pins.
NA#	I	NEXT ADDRESS is used to request address pipelining.
READY#	I	BUS READY terminates the bus cycle.
BS16#	I	BUS SIZE 16 input allows direct connection of 32-bit and 16-bit data buses.
HOLD	I	BUS HOLD REQUEST input allows another bus master to request control of the local bus.

1.1 PIN DESCRIPTION TABLE (Continued)

Symbol	Type	Name and Function
HLDA	O	BUS HOLD ACKNOWLEDGE output indicates that the 386 DX has surrendered control of its local bus to another bus master.
BUSY#	I	BUSY signals a busy condition from a processor extension.
ERROR#	I	ERROR signals an error condition from a processor extension.
PEREQ	I	PROCESSOR EXTENSION REQUEST indicates that the processor extension has data to be transferred by the 386 DX.
INTR	I	INTERRUPT REQUEST is a maskable input that signals the 386 DX to suspend execution of the current program and execute an interrupt acknowledge function.
NMI	I	NON-MASKABLE INTERRUPT REQUEST is a non-maskable input that signals the 386 DX to suspend execution of the current program and execute an interrupt acknowledge function.
RESET	I	RESET suspends any operation in progress and places the 386 DX in a known reset state. See Interrupt Signals for additional information.
N/C	—	NO CONNECT should always remain unconnected. Connection of a N/C pin may cause the processor to malfunction or be incompatible with future steppings of the 386 DX.
V _{CC}	I	SYSTEM POWER provides the +5V nominal D.C. supply input.
V _{SS}	I	SYSTEM GROUND provides 0V connection from which all inputs and outputs are measured.

2. BASE ARCHITECTURE

2.1 INTRODUCTION

The 386 DX consists of a central processing unit, a memory management unit and a bus interface.

The central processing unit consists of the execution unit and instruction unit. The execution unit contains the eight 32-bit general purpose registers which are used for both address calculation, data operations and a 64-bit barrel shifter used to speed shift, rotate, multiply, and divide operations. The multiply and divide logic uses a 1-bit per cycle algorithm. The multiply algorithm stops the iteration when the most significant bits of the multiplier are all zero. This allows typical 32-bit multiplies to be executed in under one microsecond. The instruction unit decodes the instruction opcodes and stores them in the decoded instruction queue for immediate use by the execution unit.

The memory management unit (MMU) consists of a segmentation unit and a paging unit. Segmentation allows the managing of the logical address space by providing an extra addressing component, one that allows easy code and data relocatability, and efficient sharing. The paging mechanism operates beneath and is transparent to the segmentation process, to allow management of the physical address space. Each segment is divided into one or more 4K byte pages. To implement a virtual memory system, the 386 DX supports full restartability for all page and segment faults.

Memory is organized into one or more variable length segments, each up to four gigabytes in size. A given region of the linear address space, a segment, can have attributes associated with it. These attributes include its location, size, type (i.e. stack, code or data), and protection characteristics. Each task on an 386 DX can have a maximum of 16,381 segments of up to four gigabytes each, thus providing 64 terabytes (trillion bytes) of virtual memory to each task.

The segmentation unit provides four-levels of protection for isolating and protecting applications and the operating system from each other. The hardware enforced protection allows the design of systems with a high degree of integrity.

The 386 DX has two modes of operation: Real Address Mode (Real Mode), and Protected Virtual Address Mode (Protected Mode). In Real Mode the

386 DX operates as a very fast 8086, but with 32-bit extensions if desired. Real Mode is required primarily to setup the processor for Protected Mode operation. Protected Mode provides access to the sophisticated memory management, paging and privilege capabilities of the processor.

Within Protected Mode, software can perform a task switch to enter into tasks designated as Virtual 8086 Mode tasks. Each such task behaves with 8086 semantics, thus allowing 8086 software (an application program, or an entire operating system) to execute. The Virtual 8086 tasks can be isolated and protected from one another and the host 386 DX operating system, by the use of paging, and the I/O Permission Bitmap.

Finally, to facilitate high performance system hardware designs, the 386 DX bus interface offers address pipelining, dynamic data bus sizing, and direct Byte Enable signals for each byte of the data bus. These hardware features are described fully beginning in Section 5.

2.2 REGISTER OVERVIEW

The 386 DX has 32 register resources in the following categories:

- General Purpose Registers
- Segment Registers
- Instruction Pointer and Flags
- Control Registers
- System Address Registers
- Debug Registers
- Test Registers.

The registers are a superset of the 8086, 80186 and 80286 registers, so all 16-bit 8086, 80186 and 80286 registers are contained within the 32-bit 386 DX.

Figure 2-1 shows all of 386 DX base architecture registers, which include the general address and data registers, the instruction pointer, and the flags register. The contents of these registers are task-specific, so these registers are automatically loaded with a new context upon a task switch operation.

The base architecture also includes six directly accessible segments, each up to 4 Gbytes in size. The segments are indicated by the selector values placed in 386 DX segment registers of Figure 2-1. Various selector values can be loaded as a program executes, if desired.

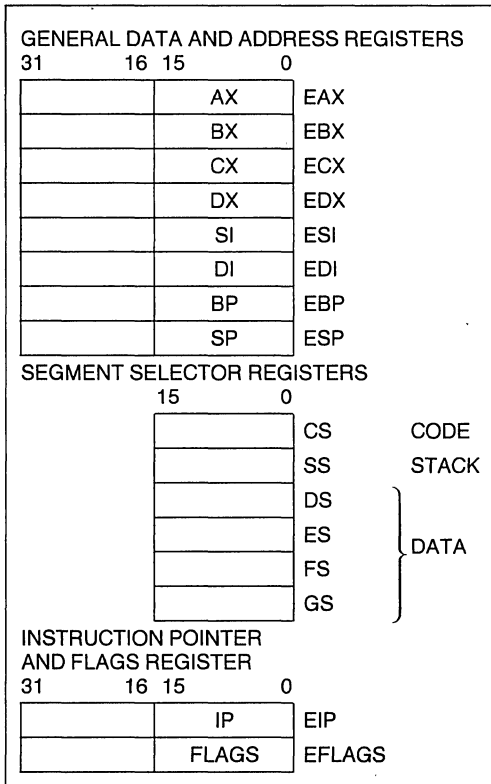


Figure 2-1. 386™ DX Base Architecture Registers

The selectors are also task-specific, so the segment registers are automatically loaded with new context upon a task switch operation.

The other types of registers, Control, System Address, Debug, and Test, are primarily used by system software.

2.3 REGISTER DESCRIPTIONS

2.3.1 General Purpose Registers

General Purpose Registers: The eight general purpose registers of 32 bits hold data or address quantities. The general registers, Figure 2-2, support data operands of 1, 8, 16, 32 and 64 bits, and bit fields of 1 to 32 bits. They support address operands of 16 and 32 bits. The 32-bit registers are named EAX, EBX, ECX, EDX, ESI, EDI, EBP, and ESP.

The least significant 16 bits of the registers can be accessed separately. This is done by using the 16-bit names of the registers AX, BX, CX, DX, SI, DI,

BP, and SP. When accessed as a 16-bit operand, the upper 16 bits of the register are neither used nor changed.

Finally 8-bit operations can individually access the lowest byte (bits 0–7) and the higher byte (bits 8–15) of general purpose registers AX, BX, CX and DX. The lowest bytes are named AL, BL, CL and DL, respectively. The higher bytes are named AH, BH, CH and DH, respectively. The individual byte accessibility offers additional flexibility for data operations, but is not used for effective address calculation.

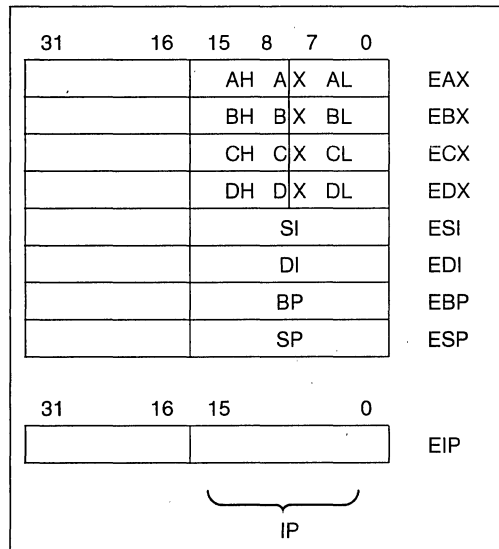


Figure 2-2. General Registers and Instruction Pointer

2.3.2 Instruction Pointer

The instruction pointer, Figure 2-2, is a 32-bit register named EIP. EIP holds the offset of the next instruction to be executed. The offset is always relative to the base of the code segment (CS). The lower 16 bits (bits 0–15) of EIP contain the 16-bit instruction pointer named IP, which is used by 16-bit addressing.

2.3.3 Flags Register

The Flags Register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS, shown in Figure 2-3, control certain operations and indicate status of the 386 DX. The lower 16 bits (bit 0–15) of EFLAGS contain the 16-bit flag register named FLAGS, which is most useful when executing 8086 and 80286 code.

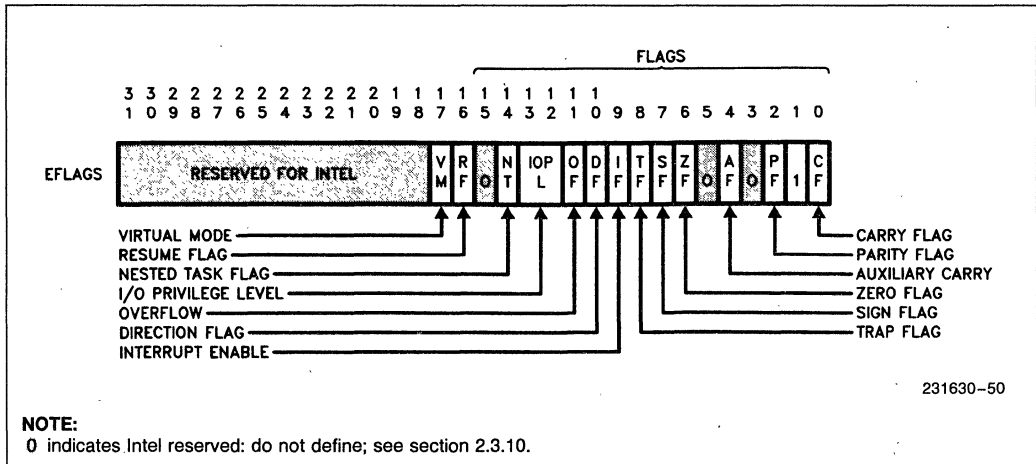


Figure 2-3. Flags Register

VM (Virtual 8086 Mode, bit 17)

The VM bit provides Virtual 8086 Mode within Protected Mode. If set while the 386 DX is in Protected Mode, the 386 DX will switch to Virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes. The VM bit can be set only in Protected Mode, by the IRET instruction (if current privilege level = 0) and by task switches at any privilege level. The VM bit is unaffected by POPF. PUSHF always pushes a 0 in this bit, even if executing in virtual 8086 Mode. The EFLAGS image pushed during interrupt processing or saved during task switches will contain a 1 in this bit if the interrupted code was executing as a Virtual 8086 Task.

RF (Resume Flag, bit 16)

The RF flag is used in conjunction with the debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. When RF is set, it causes any debug fault to be ignored on the next instruction. RF is then automatically reset at the successful completion of every instruction (no faults are signalled) except the IRET instruction, the POPF instruction, (and JMP, CALL, and INT instructions causing a task switch). These instructions set RF to the value specified by the memory image. For example, at the end of the breakpoint service routine, the IRET

instruction can pop an EFLAG image having the RF bit set and resume the program's execution at the breakpoint address without generating another breakpoint fault on the same location.

NT (Nested Task, bit 14)

This flag applies to Protected Mode. NT is set to indicate that the execution of this task is nested within another task. If set, it indicates that the current nested task's Task State Segment (TSS) has a valid back link to the previous task's TSS. This bit is set or reset by control transfers to other tasks. The value of NT in EFLAGS is tested by the IRET instruction to determine whether to do an inter-task return or an intra-task return. A POPF or an IRET instruction will affect the setting of this bit according to the image popped, at any privilege level.

IOPL (Input/Output Privilege Level, bits 12-13)

This two-bit field applies to Protected Mode. IOPL indicates the numerically maximum CPL (current privilege level) value permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O Permission Bitmap. It also indicates the maximum CPL value allowing alteration of the IF (INTR Enable Flag) bit when new values are popped into the EFLAG register. POPF and IRET instruction can alter the IOPL field when executed at CPL = 0. Task switches can always alter the IOPL field, when the new flag image is loaded from the incoming task's TSS.

- OF (Overflow Flag, bit 11)
OF is set if the operation resulted in a signed overflow. Signed overflow occurs when the operation resulted in carry/borrow **into** the sign bit (high-order bit) of the result but did not result in a carry/borrow **out of** the high-order bit, or vice-versa. For 8/16/32 bit operations, OF is set according to overflow at bit 7/15/31, respectively.
- DF (Direction Flag, bit 10)
DF defines whether ESI and/or EDI registers postdecrement or postincrement during the string instructions. Postincrement occurs if DF is reset. Postdecrement occurs if DF is set.
- IF (INTR Enable Flag, bit 9)
The IF flag, when set, allows recognition of external interrupts signalled on the INTR pin. When IF is reset, external interrupts signalled on the INTR are not recognized. IOPL indicates the maximum CPL value allowing alteration of the IF bit when new values are popped into EFLAGS or FLAGS.
- TF (Trap Enable Flag, bit 8)
TF controls the generation of exception 1 trap when single-stepping through code. When TF is set, the 386 DX generates an exception 1 trap after the next instruction is executed. When TF is reset, exception 1 traps occur only as a function of the breakpoint addresses loaded into debug registers DR0-DR3.
- SF (Sign Flag, bit 7)
SF is set if the high-order bit of the result is set, it is reset otherwise. For 8-, 16-, 32-bit operations, SF reflects the state of bit 7, 15, 31 respectively.

- ZF (Zero Flag, bit 6)
ZF is set if all bits of the result are 0. Otherwise it is reset.
- AF (Auxiliary Carry Flag, bit 4)
The Auxiliary Flag is used to simplify the addition and subtraction of packed BCD quantities. AF is set if the operation resulted in a carry out of bit 3 (addition) or a borrow into bit 3 (subtraction). Otherwise AF is reset. AF is affected by carry out of, or borrow into bit 3 only, regardless of overall operand length: 8, 16 or 32 bits.
- PF (Parity Flags, bit 2)
PF is set if the low-order eight bits of the operation contains an even number of "1's" (even parity). PF is reset if the low-order eight bits have odd parity. PF is a function of only the low-order eight bits, regardless of operand size.
- CF (Carry Flag, bit 0)
CF is set if the operation resulted in a carry out of (addition), or a borrow into (subtraction) the high-order bit. Otherwise CF is reset. For 8-, 16- or 32-bit operations, CF is set according to carry/borrow at bit 7, 15 or 31, respectively.

Note in these descriptions, "set" means "set to 1," and "reset" means "reset to 0."

2.3.4 Segment Registers

Six 16-bit segment registers hold segment selector values identifying the currently addressable memory segments. Segment registers are shown in Figure 2-4. In Protected Mode, each segment may range in size from one byte up to the entire linear and physi-

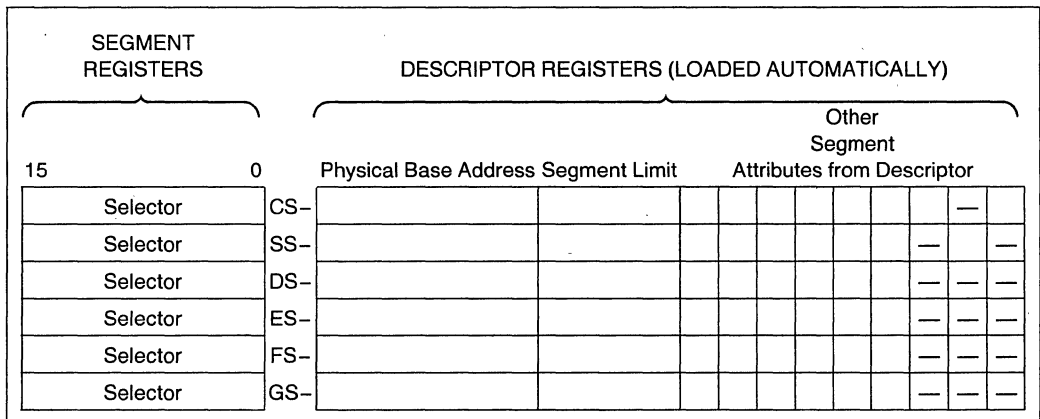


Figure 2-4. 386™ DX Segment Registers, and Associated Descriptor Registers

cal space of the machine, 4 Gbytes (2^{32} bytes). If a maximum sized segment is used (limit = FFFFFFFFH) it should be Dword aligned (i.e., the least two significant bits of the segment base should be zero). This will avoid a segment limit violation (exception 13) caused by the wrap around. In Real Address Mode, the maximum segment size is fixed at 64 Kbytes (2^{16} bytes).

The six segments addressable at any given moment are defined by the segment registers CS, SS, DS, ES, FS and GS. The selector in CS indicates the current code segment; the selector in SS indicates the current stack segment; the selectors in DS, ES, FS and GS indicate the current data segments.

2.3.5 Segment Descriptor Registers

The segment descriptor registers are not programmer visible, yet it is very useful to understand their content. Inside the 386 DX, a descriptor register (programmer invisible) is associated with each programmer-visible segment register, as shown by Figure 2-4. Each descriptor register holds a 32-bit segment base address, a 32-bit segment limit, and the other necessary segment attributes.

When a selector value is loaded into a segment register, the associated descriptor register is automatically updated with the correct information. In Real Address Mode, only the base address is updated directly (by shifting the selector value four bits to the left), since the segment maximum limit and attributes are fixed in Real Mode. In Protected Mode, the base address, the limit, and the attributes are all updated per the contents of the segment descriptor indexed by the selector.

Whenever a memory reference occurs, the segment descriptor register associated with the segment being used is automatically involved with the memory reference. The 32-bit segment base address becomes a component of the linear address calcula-

tion, the 32-bit limit is used for the limit-check operation, and the attributes are checked against the type of memory reference requested.

2.3.6 Control Registers

The 386 DX has three control registers of 32 bits, CR0, CR2 and CR3, to hold machine state of a global nature (not specific to an individual task). These registers, along with System Address Registers described in the next section, hold machine state that affects all tasks in the system. To access the Control Registers, load and store instructions are defined.

CR0: Machine Control Register (includes 80286 Machine Status Word)

CR0, shown in Figure 2-5, contains 6 defined bits for control and status purposes. The low-order 16 bits of CR0 are also known as the Machine Status Word, MSW, for compatibility with 80286 Protected Mode. LMSW and SMSW instructions are taken as special aliases of the load and store CR0 operations, where only the low-order 16 bits of CR0 are involved. For compatibility with 80286 operating systems the 386 DX LMSW instructions work in an identical fashion to the LMSW instruction on the 80286. (i.e. It only operates on the low-order 16-bits of CR0 and it ignores the new bits in CR0.) New 386 DX operating systems should use the MOV CR0, Reg instruction.

The defined CR0 bits are described below.

PG (Paging Enable, bit 31)

the PG bit is set to enable the on-chip paging unit. It is reset to disable the on-chip paging unit.

R (reserved, bit 4)

This bit is reserved by Intel. When loading CR0 care should be taken to not alter the value of this bit.

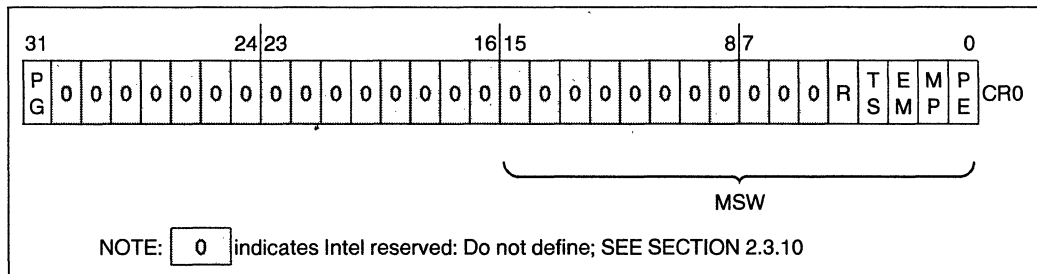


Figure 2-5. Control Register 0

TS (Task Switched, bit 3)

TS is automatically set whenever a task switch operation is performed. If TS is set, a coprocessor ESCape opcode will cause a Coprocessor Not Available trap (exception 7). The trap handler typically saves the 387 DX coprocessor context belonging to a previous task, loads the 387 DX coprocessor state belonging to the current task, and clears the TS bit before returning to the faulting coprocessor opcode.

EM (Emulate Coprocessor, bit 2)

The EMulate coprocessor bit is set to cause all coprocessor opcodes to generate a Coprocessor Not Available fault (exception 7). It is reset to allow coprocessor opcodes to be executed on an actual 387 DX coprocessor (this is the default case after reset). Note that the WAIT opcode is not affected by the EM bit setting.

MP (Monitor Coprocessor, bit 1)

The MP bit is used in conjunction with the TS bit to determine if the WAIT opcode will generate a Coprocessor Not Available fault (exception 7) when TS = 1. When both MP = 1 and TS = 1, the WAIT opcode generates a trap. Otherwise, the WAIT opcode does not generate a trap. Note that TS is automatically set whenever a task switch operation is performed.

PE (Protection Enable, bit 0)

The PE bit is set to enable the Protected Mode. If PE is reset, the processor operates again in Real Mode. PE may be set by loading MSW or CR0. PE can be reset only by a load into CR0. Resetting the PE bit is typically part of a longer instruction sequence needed for proper transition from Protected Mode to Real Mode. Note that for strict 80286 compatibility, PE cannot be reset by the LMSW instruction.

CR1: reserved

CR1 is reserved for use in future Intel processors.

CR2: Page Fault Linear Address

CR2, shown in Figure 2-6, holds the 32-bit linear address that caused the last page fault detected. The

error code pushed onto the page fault handler's stack when it is invoked provides additional status information on this page fault.

CR3: Page Directory Base Address

CR3, shown in Figure 2-6, contains the physical base address of the page directory table. The 386 DX page directory table is always page-aligned (4 Kbyte-aligned). Therefore the lowest twelve bits of CR3 are ignored when written and they store as undefined.

A task switch through a TSS which **changes** the value in CR3, or an explicit load into CR3 with any value, will invalidate all cached page table entries in the paging unit cache. Note that if the value in CR3 does not change during the task switch, the cached page table entries are not flushed.

2.3.7 System Address Registers

Four special registers are defined to reference the tables or segments supported by the 80286 CPU and 386 DX protection model. These tables or segments are:

- GDT (Global Descriptor Table),
- IDT (Interrupt Descriptor Table),
- LDT (Local Descriptor Table),
- TSS (Task State Segment).

The addresses of these tables and segments are stored in special registers, the System Address and System Segment Registers illustrated in Figure 2-7. These registers are named GDTR, IDTR, LDTR and TR, respectively. Section 4 **Protected Mode Architecture** describes the use of these registers.

GDTR and IDTR

These registers hold the 32-bit linear base address and 16-bit limit of the GDT and IDT, respectively.

The GDT and IDT segments, since they are global to all tasks in the system, are defined by 32-bit linear addresses (subject to page translation if paging is enabled) and 16-bit limit values.

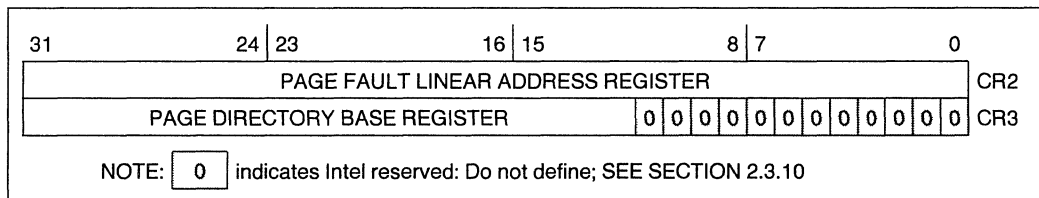


Figure 2-6. Control Registers 2 and 3

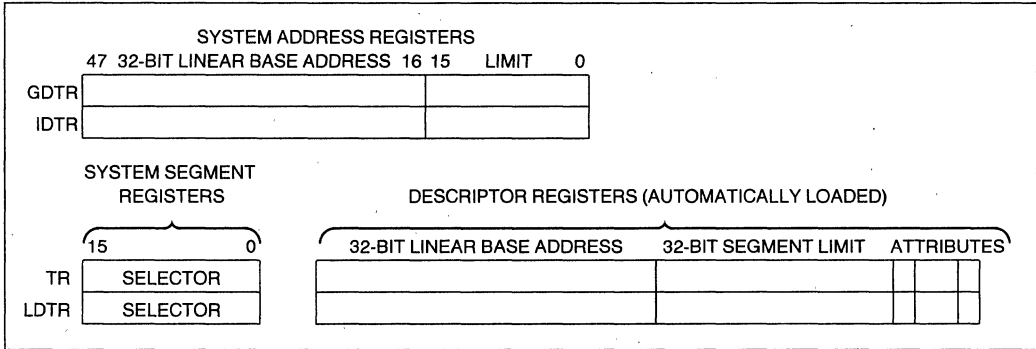


Figure 2-7. System Address and System Segment Registers

LDTR and TR

These registers hold the 16-bit selector for the LDT descriptor and the TSS descriptor, respectively.

The LDT and TSS segments, since they are task-specific segments, are defined by selector values stored in the system segment registers. Note that a segment descriptor register (programmer-invisible) is associated with each system segment register.

2.3.8 Debug and Test Registers

Debug Registers: The six programmer accessible debug registers provide on-chip support for debugging. Debug Registers DR0–3 specify the four linear breakpoints. The Debug Control Register DR7 is used to set the breakpoints and the Debug Status Register DR6, displays the current state of the breakpoints. The use of the debug registers is described in section 2.12 **Debugging support**.

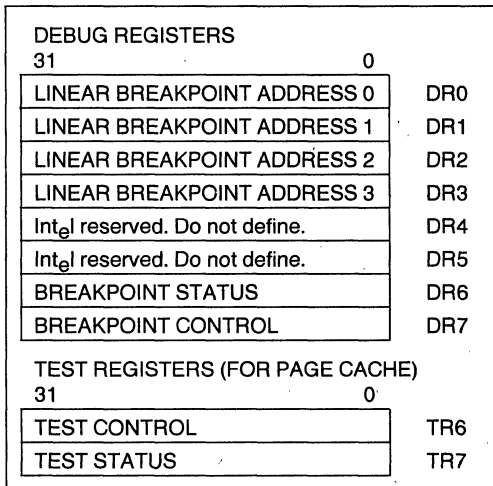


Figure 2-8. Debug and Test Registers

Test Registers: Two registers are used to control the testing of the RAM/CAM (Content Addressable Memories) in the Translation Lookaside Buffer portion of the 386 DX. TR6 is the command test register, and TR7 is the data register which contains the data of the Translation Lookaside buffer test. Their use is discussed in section 2.11 **Testability**.

Figure 2-8 shows the Debug and Test registers.

2.3.9 Register Accessibility

There are a few differences regarding the accessibility of the registers in Real and Protected Mode. Table 2-1 summarizes these differences. See Section 4 **Protected Mode Architecture** for further details.

2.3.10 Compatibility

**VERY IMPORTANT NOTE:
 COMPATIBILITY WITH FUTURE PROCESSORS**

In the preceding register descriptions, note certain 386 DX register bits are Intel reserved. When reserved bits are called out, treat them as fully undefined. This is essential for your software compatibility with future processors! Follow the guidelines below:

- 1) Do not depend on the states of any undefined bits when testing the values of defined register bits. Mask them out when testing.
- 2) Do not depend on the states of any undefined bits when storing them to memory or another register.
- 3) Do not depend on the ability to retain information written into any undefined bits.
- 4) When loading registers always load the undefined bits as zeros.

Table 2-1. Register Usage

Register	Use in Real Mode		Use in Protected Mode		Use in Virtual 8086 Mode	
	Load	Store	Load	Store	Load	Store
General Registers	Yes	Yes	Yes	Yes	Yes	Yes
Segment Registers	Yes	Yes	Yes	Yes	Yes	Yes
Flag Register	Yes	Yes	Yes	Yes	IOPL*	IOPL*
Control Registers	Yes	Yes	PL = 0	PL = 0	No	Yes
GDTR	Yes	Yes	PL = 0	Yes	No	Yes
IDTR	Yes	Yes	PL = 0	Yes	No	Yes
LDTR	No	No	PL = 0	Yes	No	No
TR	No	No	PL = 0	Yes	No	No
Debug Control	Yes	Yes	PL = 0	PL = 0	No	No
Test Registers	Yes	Yes	PL = 0	PL = 0	No	No

NOTES:

PL = 0: The registers can be accessed only when the current privilege level is zero.

*IOPL: The PUSHF and POPF instructions are made I/O Privilege Level sensitive in Virtual 8086 Mode.

5) However, registers which have been previously stored may be reloaded without masking.

Depending upon the values of undefined register bits will make your software dependent upon the unspecified 386 DX handling of these bits. Depending on undefined values risks making your software incompatible with future processors that define usages for the 386 DX-undefined bits. **AVOID ANY SOFTWARE DEPENDENCE UPON THE STATE OF UNDEFINED 386 DX REGISTER BITS.**

2.4 INSTRUCTION SET

2.4.1 Instruction Set Overview

The instruction set is divided into nine categories of operations:

- Data Transfer
- Arithmetic
- Shift/Rotate
- String Manipulation
- Bit Manipulation
- Control Transfer
- High Level Language Support
- Operating System Support
- Processor Control

These 386 DX instructions are listed in Table 2-2.

All 386 DX instructions operate on either 0, 1, 2, or 3 operands; where an operand resides in a register, in the instruction itself, or in memory. Most zero operand instructions (e.g. CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the 386 DX has a 16-byte instruction queue, an average of 5 instructions will be prefetched. The use of two operands permits the following types of common instructions:

- Register to Register
- Memory to Register
- Immediate to Register
- Register to Memory
- Immediate to Memory.

The operands can be either 8, 16, or 32 bits long. As a general rule, when executing code written for the 386 DX (32-bit code), operands are 8 or 32 bits; when executing existing 80286 or 8086 code (16-bit code), operands are 8 or 16 bits. Prefixes can be added to all instructions which override the default length of the operands, (i.e. use 32-bit operands for 16-bit code, or 16-bit operands for 32-bit code).

For a more elaborate description of the instruction set, refer to the "386 DX Programmer's Reference Manual."

2.4.2 386™ DX Instructions
Table 2-2a. Data Transfer

GENERAL PURPOSE	
MOV	Move operand
PUSH	Push operand onto stack
POP	Pop operand off stack
PUSHA	Push all registers on stack
POPA	Pop all registers off stack
XCHG	Exchange Operand, Register
XLAT	Translate
CONVERSION	
MOVZX	Move byte or Word, Dword, with zero extension
MOVSX	Move byte or Word, Dword, sign extended
CBW	Convert byte to Word, or Word to Dword
CWD	Convert Word to DWORD
CWDE	Convert Word to DWORD extended
CDQ	Convert DWORD to QWORD
INPUT/OUTPUT	
IN	Input operand from I/O space
OUT	Output operand to I/O space
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer into D segment register
LES	Load pointer into E segment register
LFS	Load pointer into F segment register
LGS	Load pointer into G segment register
LSS	Load pointer into S (Stack) segment register
FLAG MANIPULATION	
LAHF	Load A register from Flags
SAHF	Store A register in Flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
PUSHFD	Push EFlags onto stack
POPFD	Pop EFlags off stack
CLC	Clear Carry Flag
CLD	Clear Direction Flag
CMC	Complement Carry Flag
STC	Set Carry Flag
STD	Set Direction Flag

Table 2-2b. Arithmetic Instructions

ADDITION	
ADD	Add operands
ADC	Add with carry
INC	Increment operand by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract operands
SBB	Subtract with borrow
DEC	Decrement operand by 1
NEG	Negate operand
CMP	Compare operands
DAS	Decimal adjust for subtraction
AAS	ASCII Adjust for subtraction
MULTIPLICATION	
MUL	Multiply Double/Single Precision
IMUL	Integer multiply
AAM	ASCII adjust after multiply
DIVISION	
DIV	Divide unsigned
IDIV	Integer Divide
AAD	ASCII adjust before division

Table 2-2c. String Instructions

MOVS	Move byte or Word, Dword string
INS	Input string from I/O space
OUTS	Output string to I/O space
CMPS	Compare byte or Word, Dword string
SCAS	Scan Byte or Word, Dword string
LODS	Load byte or Word, Dword string
STOS	Store byte or Word, Dword string
REP	Repeat
REPE/ REPZ	Repeat while equal/zero
RENE/ REPNZ	Repeat while not equal/not zero

Table 2-2d. Logical Instructions

LOGICALS	
NOT	"NOT" operands
AND	"AND" operands
OR	"Inclusive OR" operands
XOR	"Exclusive OR" operands
TEST	"Test" operands

Table 2-2d. Logical Instructions (Continued)

SHIFTS	
SHL/SHR	Shift logical left or right
SAL/SAR	Shift arithmetic left or right
SHLD/SHRD	Double shift left or right
ROTATES	
ROL/ROR	Rotate left/right
RCL/RCR	Rotate through carry left/right

Table 2-2e. Bit Manipulation Instructions

SINGLE BIT INSTRUCTIONS	
BT	Bit Test
BTS	Bit Test and Set
BTR	Bit Test and Reset
BTC	Bit Test and Complement
BSF	Bit Scan Forward
BSR	Bit Scan Reverse

Table 2-2f. Program Control Instructions

CONDITIONAL TRANSFERS	
SETCC	Set byte equal to condition code
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign
JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if Sign

Table 2-2f. Program Control Instructions (Continued)

UNCONDITIONAL TRANSFERS	
CALL	Call procedure/task
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCZX	JUMP if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Return from Interrupt/Task
CLI	Clear interrupt Enable
STI	Set Interrupt Enable

Table 2-2g. High Level Language Instructions

BOUND	Check Array Bounds
ENTER	Setup Parameter Block for Entering Procedure
LEAVE	Leave Procedure

Table 2-2h. Protection Model

SGDT	Store Global Descriptor Table
SIDT	Store Interrupt Descriptor Table
STR	Store Task Register
SLDT	Store Local Descriptor Table
LGDT	Load Global Descriptor Table
LIDT	Load Interrupt Descriptor Table
LTR	Load Task Register
LLDT	Load Local Descriptor Table
ARPL	Adjust Requested Privilege Level
LAR	Load Access Rights
LSL	Load Segment Limit
VERR/VERW	Verify Segment for Reading or Writing
LMSW	Load Machine Status Word (lower 16 bits of CR0)
SMSW	Store Machine Status Word

Table 2-2i. Processor Control Instructions

HLT	Halt
WAIT	Wait until BUSY # negated
ESC	Escape
LOCK	Lock Bus

2.5 ADDRESSING MODES

2.5.1 Addressing Modes Overview

The 386 DX provides a total of 11 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

2.5.2 Register and Immediate Modes

Two of the addressing modes provide for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8-, 16- or 32-bit general registers.

Immediate Operand Mode: The operand is included in the instruction as part of the opcode.

2.5.3 32-Bit Memory Addressing Modes

The remaining 9 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the segment base address and an effective address. The effective address is calculated by using combinations of the following four address elements:

DISPLACEMENT: An 8-, or 32-bit immediate value, following the instruction.

BASE: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area.

INDEX: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters.

SCALE: The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. Scaled index mode is especially useful for accessing arrays or structures.

Combinations of these 4 components make up the 9 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions.

The one exception is the simultaneous use of Base and Index components which requires one additional clock.

As shown in Figure 2-9, the effective address (EA) of an operand is calculated according to the following formula.

$$EA = \text{Base Reg} + (\text{Index Reg} * \text{Scaling}) + \text{Displacement}$$

Direct Mode: The operand's offset is contained as part of the instruction as an 8-, 16- or 32-bit displacement.

EXAMPLE: INC Word PTR [500]

Register Indirect Mode: A BASE register contains the address of the operand.

EXAMPLE: MOV [ECX], EDX

Based Mode: A BASE register's contents is added to a DISPLACEMENT to form the operands offset.

EXAMPLE: MOV ECX, [EAX + 24]

Index Mode: An INDEX register's contents is added to a DISPLACEMENT to form the operands offset.

EXAMPLE: ADD EAX, TABLE[ESI]

Scaled Index Mode: An INDEX register's contents is multiplied by a scaling factor which is added to a DISPLACEMENT to form the operands offset.

EXAMPLE: IMUL EBX, TABLE[ESI*4],7

Based Index Mode: The contents of a BASE register is added to the contents of an INDEX register to form the effective address of an operand.

EXAMPLE: MOV EAX, [ESI] [EBX]

Based Scaled Index Mode: The contents of an INDEX register is multiplied by a SCALING factor and the result is added to the contents of a BASE register to obtain the operands offset.

EXAMPLE: MOV ECX, [EDX*8] [EAX]

Based Index Mode with Displacement: The contents of an INDEX Register and a BASE register's contents and a DISPLACEMENT are all summed together to form the operand offset.

EXAMPLE: ADD EDX, [ESI] [EBP + 00FFFFFF0H]

Based Scaled Index Mode with Displacement: The contents of an INDEX register are multiplied by a SCALING factor, the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.

EXAMPLE: MOV EAX, LOCALTABLE[EDI*4] [EBP + 80]

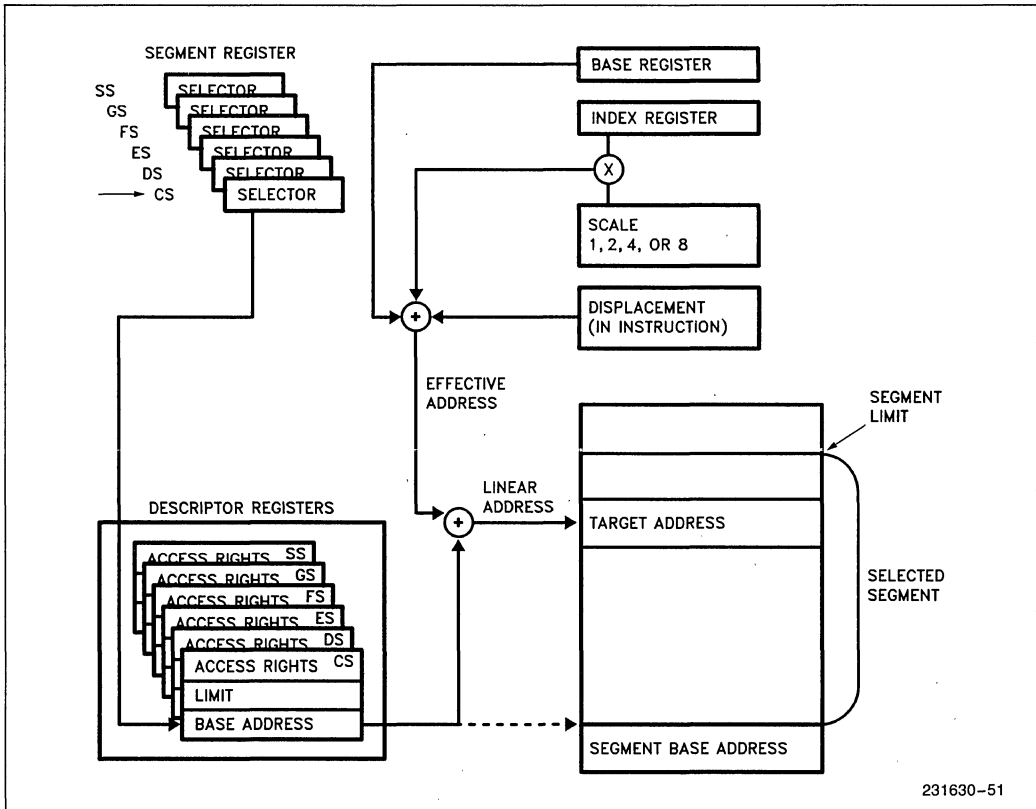


Figure 2-9. Addressing Mode Calculations

2.5.4 Differences Between 16 and 32 Bit Addresses

In order to provide software compatibility with the 80286 and the 8086, the 386 DX can execute 16-bit instructions in Real and Protected Modes. The processor determines the size of the instructions it is executing by examining the D bit in the CS segment Descriptor. If the D bit is 0 then all operand lengths and effective addresses are assumed to be 16 bits long. If the D bit is 1 then the default length for operands and addresses is 32 bits. In Real Mode the default size for operands and addresses is 16-bits.

Regardless of the default precision of the operands or addresses, the 386 DX is able to execute either 16 or 32-bit instructions. This is specified via the use of override prefixes. Two prefixes, the **Operand Size Prefix** and the **Address Length Prefix**, override the value of the D bit on an individual instruction basis. These prefixes are automatically added by Intel assemblers.

Example: The processor is executing in Real Mode and the programmer needs to access the EAX registers. The assembler code for this might be `MOV EAX, 32-bit MEMORYOP`, ASM386 Macro Assembler automatically determines that an Operand Size Prefix is needed and generates it.

Example: The D bit is 0, and the programmer wishes to use Scaled Index addressing mode to access an array. The Address Length Prefix allows the use of `MOV DX, TABLE[ESI*2]`. The assembler uses an Address Length Prefix since, with D=0, the default addressing mode is 16-bits.

Example: The D bit is 1, and the program wants to store a 16-bit quantity. The Operand Length Prefix is used to specify only a 16-bit value; `MOV MEM16, DX`.

Table 2-3. BASE and INDEX Registers for 16- and 32-Bit Addresses

	16-Bit Addressing	32-Bit Addressing
BASE REGISTER	BX,BP	Any 32-bit GP Register
INDEX REGISTER	SI,DI	Any 32-bit GP Register Except ESP
SCALE FACTOR	none	1, 2, 4, 8
DISPLACEMENT	0, 8, 16 bits	0, 8, 32 bits

The OPERAND LENGTH and Address Length Prefixes can be applied separately or in combination to any instruction. The Address Length Prefix does not allow addresses over 64K bytes to be accessed in Real Mode. A memory address which exceeds FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional 386 DX addressing modes.

When executing 32-bit code, the 386 DX uses either 8-, or 32-bit displacements, and any register can be used as base or index registers. When executing 16-bit code, the displacements are either 8, or 16 bits, and the base and index register conform to the 80286 model. Table 2-3 illustrates the differences.

2.6 DATA TYPES

The 386 DX supports all of the data types commonly used in high level languages:

Bit: A single bit quantity.

Bit Field: A group of up to 32 contiguous bits, which spans a maximum of four bytes.

Bit String: A set of contiguous bits, on the 386 DX bit strings can be up to 4 gigabits long.

Byte: A signed 8-bit quantity.

Unsigned Byte: An unsigned 8-bit quantity.

Integer (Word): A signed 16-bit quantity.

Long Integer (Double Word): A signed 32-bit quantity. All operations assume a 2's complement representation.

Unsigned Integer (Word): An unsigned 16-bit quantity.

Unsigned Long Integer (Double Word): An unsigned 32-bit quantity.

Signed Quad Word: A signed 64-bit quantity.

Unsigned Quad Word: An unsigned 64-bit quantity.

Offset: A 16- or 32-bit offset only quantity which indirectly references another memory location.

Pointer: A full pointer which consists of a 16-bit segment selector and either a 16- or 32-bit offset.

Char: A byte representation of an ASCII Alphanumeric or control character.

String: A contiguous sequence of bytes, words or dwords. A string may contain between 1 byte and 4 Gbytes.

BCD: A byte (unpacked) representation of decimal digits 0–9.

Packed BCD: A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble.

When the 386 DX is coupled with a 387 DX Numerics Coprocessor then the following common Floating Point types are supported.

Floating Point: A signed 32-, 64-, or 80-bit real number representation. Floating point numbers are supported by the 387 DX numerics coprocessor.

Figure 2-10 illustrates the data types supported by the 386 DX and the 387 DX numerics coprocessor.

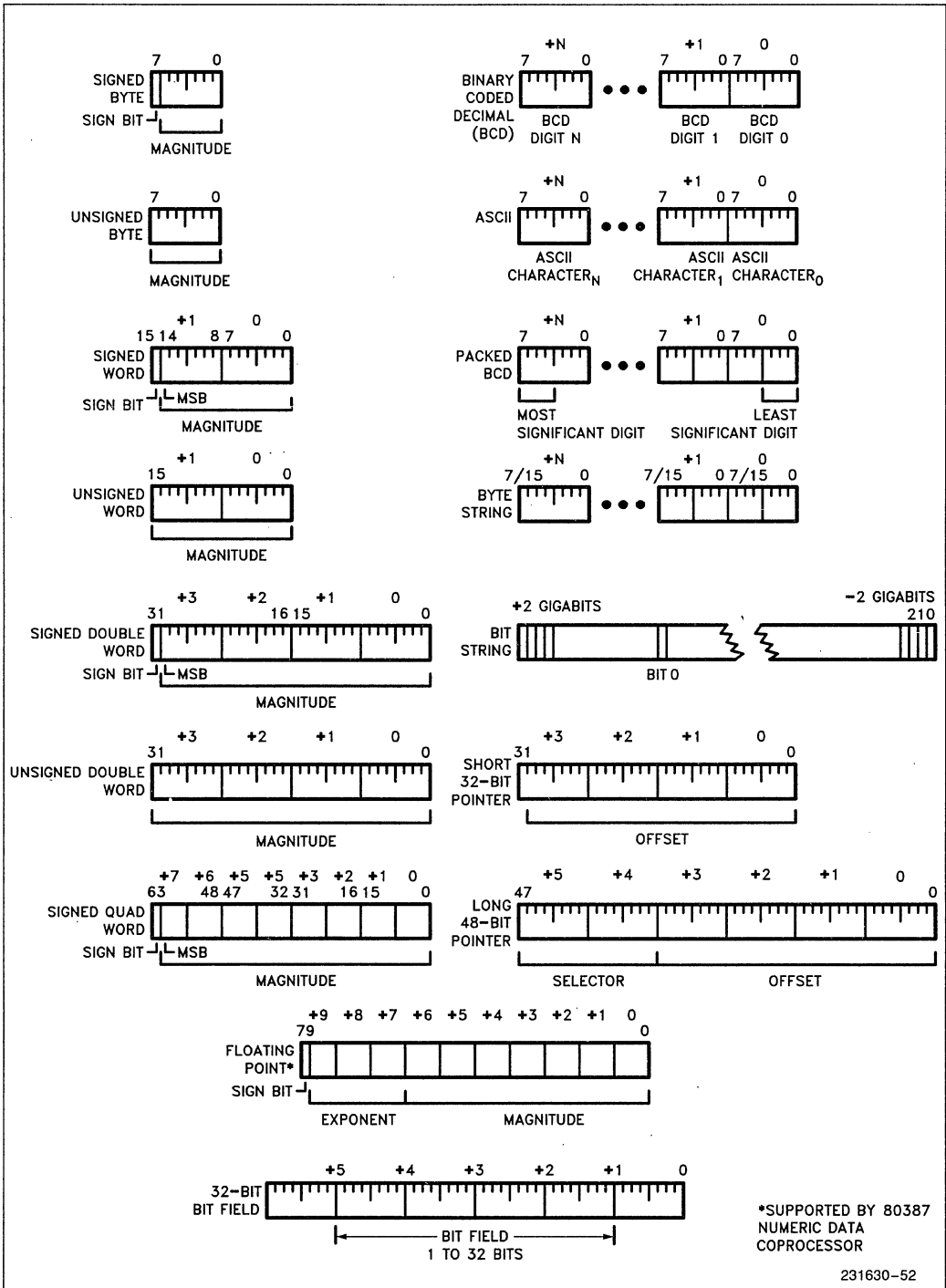


Figure 2-10. 386™ DX Supported Data Types

2.7 MEMORY ORGANIZATION

2.7.1 Introduction

Memory on the 386 DX is divided up into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address, the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address, the high-order byte at the highest address. The address of a word or dword is the byte address of the low-order byte.

In addition to these basic data types, the 386 DX supports two larger units of memory: pages and segments. Memory can be divided up into one or more variable length segments, which can be swapped to disk or shared between programs. Memory can also be organized into one or more 4K byte pages. Finally, both segmentation and paging can be combined, gaining the advantages of both systems. The 386 DX supports both pages and segments in order to provide maximum flexibility to the system designer. Segmentation and paging are complementary. Segmentation is useful for organizing memory in logical modules, and as such is a tool for the application programmer, while pages are useful for the system programmer for managing the physical memory of a system.

2.7.2 Address Spaces

The 386 DX has three distinct address spaces: **logical**, **linear**, and **physical**. A **logical** address

(also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, DISPLACEMENT) discussed in section 2.5.3 **Memory Addressing Modes** into an effective address. Since each task on 386 DX has a maximum of 16K ($2^{14} - 1$) selectors, and offsets can be 4 gigabytes, (2^{32} bits) this gives a total of 2^{46} bits or 64 terabytes of **logical** address space per task. The programmer sees this virtual address space.

The segmentation unit translates the **logical** address space into a 32-bit **linear** address space. If the paging unit is not enabled then the 32-bit **linear** address corresponds to the **physical** address. The paging unit translates the **linear** address space into the **physical** address space. The **physical address** is what appears on the address pins.

The primary difference between Real Mode and Protected Mode is how the segmentation unit performs the translation of the **logical** address into the **linear** address. In Real Mode, the segmentation unit shifts the selector left four bits and adds the result to the offset to form the **linear** address. While in Protected Mode every selector has a **linear** base address associated with it. The **linear base** address is stored in one of two operating system tables (i.e. the Local Descriptor Table or Global Descriptor Table). The selector's **linear base** address is added to the offset to form the final **linear** address.

Figure 2-11 shows the relationship between the various address spaces.

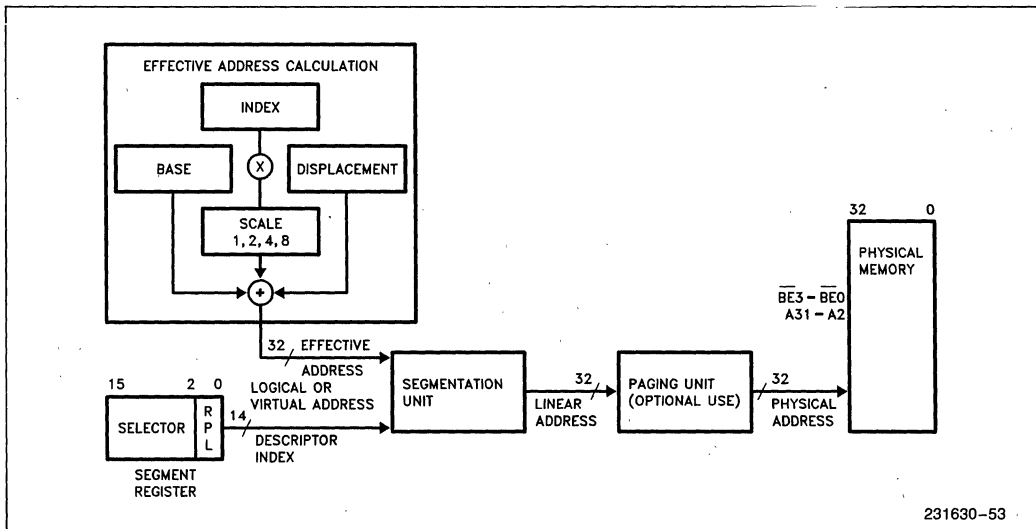


Figure 2-11. Address Translation

2.7.3 Segment Register Usage

The main data structure used to organize memory is the segment. On the 386 DX, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments: code and data, the segments are of variable size and can be as small as 1 byte or as large as 4 gigabytes (2^{32} bytes).

In order to provide compact instruction encoding, and increase processor performance, instructions do not need to explicitly specify which segment register is used. A default segment register is automatically chosen according to the rules of Table 2-4 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register; Stack references use the SS register and Instruction fetches use the CS register. The contents of the Instruction Pointer provides the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2-4. The override prefixes also allow the use of the ES, FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero and create a system with a four gigabyte linear address space. This creates a system where the virtual address space is the same as the linear address space. Further details of segmentation are discussed in section 4.1.

2.8 I/O SPACE

The 386 DX has two distinct physical address spaces: Memory and I/O. Generally, peripherals are placed in I/O space although the 386 DX also supports memory-mapped peripherals. The I/O space consists of 64K bytes, it can be divided into 64K 8-bit ports, 32K 16-bit ports, or 16K 32-bit ports, or any combination of ports which add up to less than 64K bytes. The 64K I/O address space refers to physical memory rather than linear address since I/O instructions do not go through the segmentation or paging hardware. The M/IO# pin acts as an additional address line thus allowing the system designer to easily determine which address space the processor is accessing.

Table 2-4. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA Instructions	SS	None
Source of POP, POPA, POPF, IRET, RET instructions	SS	None
Destination of STOS, MOVS, REP STOS, REP MOVS Instructions (DI is Base Register)	ES	None
Other Data References, with Effective Address Using Base Register of:		
[EAX]	DS	DS,CS,SS,ES,FS,GS
[EBX]	DS	DS,CS,SS,ES,FS,GS
[ECX]	DS	DS,CS,SS,ES,FS,GS
[EDX]	DS	DS,CS,SS,ES,FS,GS
[ESI]	DS	DS,CS,SS,ES,FS,GS
[EDI]	DS	DS,CS,SS,ES,FS,GS
[EBP]	SS	DS,CS,SS,ES,FS,GS
[ESP]	SS	DS,CS,SS,ES,FS,GS

The I/O ports are accessed via the IN and OUT I/O instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8- and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/IO# pin to be driven low.

I/O port addresses 00F8H through 00FFH are reserved for use by Intel.

2.9 INTERRUPTS

2.9.1 Interrupts and Exceptions

Interrupts and exceptions alter the normal program flow, in order to handle external events, to report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT N instruction, the processor treats software interrupts as exceptions.

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately **after** the interrupted instruction. Sections 2.9.3 and 2.9.4 discuss the differences between Maskable and Non-Maskable interrupts.

Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is supported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. A fault would occur in a virtual memory system, when the processor referenced a page or a segment which was not present. The operating system would fetch the page or segment from disk, and then the 386 DX would restart the instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. User defined interrupts are examples of traps. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined. Aborts are used to report severe errors, such as a hardware error, or illegal values in system tables.

Thus, when an interrupt service routine has been completed, execution proceeds from the instruction

immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point at the instruction causing the exception and include any leading instruction prefixes. Table 2-5 summarizes the possible interrupts for the 386 DX and shows where the return address points.

The 386 DX has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode (see section 3.1), the vectors are 4 byte quantities, a Code Segment plus a 16-bit offset; in Protected Mode, the interrupt vectors are 8 byte quantities, which are put in an Interrupt Descriptor Table (see section 4.1). Of the 256 possible interrupts, 32 are reserved for use by Intel, the remaining 224 are free to be used by the system designer.

2.9.2 Interrupt Processing

When an interrupt occurs the following actions happen. First, the current program address and the Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the 386 DX which identifies the appropriate entry in the interrupt table. The table contains the starting address of the interrupt service routine. Then, the user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the 386 DX in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

2.9.3 Maskable Interrupt

Maskable interrupts are the most common way used by the 386 DX to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled high and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions, (REPEAT String instructions, have an "interrupt window", between memory moves, which allows interrupts during long

Table 2-5. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	YES	FAULT
Debug Exception	1	any instruction	YES	TRAP*
NMI Interrupt	2	INT 2 or NMI	NO	NMI
One Byte Interrupt	3	INT	NO	TRAP
Interrupt on Overflow	4	INTO	NO	TRAP
Array Bounds Check	5	BOUND	YES	FAULT
Invalid OP-Code	6	Any Illegal Instruction	YES	FAULT
Device Not Available	7	ESC, WAIT	YES	FAULT
Double Fault	8	Any Instruction That Can Generate an Exception		ABORT
Coprocessor Segment Overrun	9	ESC	NO	ABORT
Invalid TSS	10	JMP, CALL, IRET, INT	YES	FAULT
Segment Not Present	11	Segment Register Instructions	YES	FAULT
Stack Fault	12	Stack References	YES	FAULT
General Protection Fault	13	Any Memory Reference	YES	FAULT
Intel Reserved	15			
Page Fault	14	Any Memory Access or Code Fetch	YES	FAULT
Coprocessor Error	16	ESC, WAIT	YES	FAULT
Intel Reserved	17–31			
Two Byte Interrupt	0–255	INT n	NO	TRAP

* Some debug exceptions may report both traps on the previous instruction, and faults on the next instruction.

string moves). When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt, (one of 224 user defined interrupts). The exact nature of the interrupt sequence is discussed in section 5.

The IF bit in the EFLAG registers is reset when an interrupt is being serviced. This effectively disables servicing additional interrupts during an interrupt service routine. However, the IF may be set explicitly by the interrupt handler, to allow the nesting of interrupts. When an IRET instruction is executed the original state of the IF is restored.

2.9.4 Non-Maskable Interrupt

Non-maskable interrupts provide a method of servicing very high priority interrupts. A common example of the use of a non-maskable interrupt (NMI) would be to activate a power failure routine. When the NMI

input is pulled high it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt, no interrupt acknowledgment sequence is performed for an NMI.

While executing the NMI servicing procedure, the 386 DX will not service further NMI requests, until an interrupt return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The IF bit is cleared at the beginning of an NMI interrupt to inhibit further INTR interrupts.

2.9.5 Software Interrupts

A third type of interrupt/exception for the 386 DX is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table.

A special case of the two byte software interrupt INT n is the one byte INT 3, or breakpoint interrupt. By inserting this one byte instruction in a program, the user can set breakpoints in his program as a debugging tool.

A final type of software interrupt, is the single step interrupt. It is discussed in section 2.12.

2.9.6 Interrupt and Exception Priorities

Interrupts are externally-generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are **both** recognized at the **same** instruction boundary, the 386 DX invokes the NMI service routine first. If, after the NMI service routine has been invoked, maskable interrupts are still enabled, then the 386 DX will invoke the appropriate interrupt service routine.

Table 2-6a. 386™ DX Priority for Invoking Service Routines in Case of Simultaneous External Interrupts

1. NMI 2. INTR

Exceptions are internally-generated events. Exceptions are detected by the 386 DX if, in the course of executing an instruction, the 386 DX detects a problematic condition. The 386 DX then immediately invokes the appropriate exception service routine. The state of the 386 DX is such that the instruction causing the exception can be restarted. If the exception service routine has taken care of the problematic condition, the instruction will execute without causing the same exception.

It is possible for a single instruction to generate several exceptions (for example, transferring a single operand could generate two page faults if the operand location spans two "not present" pages). However, only one exception is generated upon each attempt to execute the instruction. Each exception service routine should correct its corresponding exception, and restart the instruction. In this manner, exceptions are serviced until the instruction executes successfully.

As the 386 DX executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2-6b. This cycle is repeated

as each instruction is executed, and occurs in parallel with instruction decoding and execution.

Table 2-6b. Sequence of Exception Checking

Consider the case of the 386 DX having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
3. Check for external NMI and INTR.
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Page Faults that prevented fetching the entire next instruction (exception 14).
6. Check for Faults decoding the next instruction (exception 6 if illegal opcode; exception 6 if in Real Mode or in Virtual 8086 Mode and attempting to execute an instruction for Protected Mode only (see 4.6.4); or exception 13 if instruction is longer than 15 bytes, or privilege violation in Protected Mode (i.e. not at IOPL or at CPL = 0).
7. If WAIT opcode, check if TS = 1 and MP = 1 (exception 7 if both are 1).
8. If ESCAPE opcode for numeric coprocessor, check if EM = 1 or TS = 1 (exception 7 if either are 1).
9. If WAIT opcode or ESCAPE opcode for numeric coprocessor, check ERROR# input signal (exception 16 if ERROR# input is asserted).
10. Check in the following order for each memory reference required by the instruction:
 - a. Check for Segmentation Faults that prevent transferring the entire memory quantity (exceptions 11, 12, 13).
 - b. Check for Page Faults that prevent transferring the entire memory quantity (exception 14).

Note that the order stated supports the concept of the paging mechanism being "underneath" the segmentation mechanism. Therefore, for any given code or data reference in memory, segmentation exceptions are generated before paging exceptions are generated.

2.9.7 Instruction Restart

The 386 DX fully supports restarting all instructions after faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10 in Table 2-6b), the 386 DX invokes the appropriate exception service routine. The 386 DX is in a state that permits restart of the instruction, for all cases but those in Table 2-6c. Note that all such cases are easily avoided by proper design of the operating system.

Table 2-6c. Conditions Preventing Instruction Restart

- A. An instruction causes a task switch to a task whose Task State Segment is **partially** "not present". (An entirely "not present" TSS is restartable.) Partially present TSS's can be avoided either by keeping the TSS's of such tasks present in memory, or by aligning TSS segments to reside entirely within a single 4K page (for TSS segments of 4K bytes or less).
- B. A coprocessor operand wraps around the top of a 64K-byte segment or a 4G-byte segment, and spans three pages, and the page holding the middle portion of the operand is "not present." This condition can be avoided by starting **at a page boundary** any segments containing coprocessor operands if the segments are approximately 64K-200 bytes or larger (i.e. large enough for wraparound of the coprocessor operand to possibly occur).

Note that these conditions are avoided by using the operating system designs mentioned in this table.

2.9.8 Double Fault

A Double Fault (exception 8) results when the processor attempts to invoke an exception service routine for the segment exceptions (10, 11, 12 or 13), but in the process of doing so, detects an exception **other than** a Page Fault (exception 14).

A Double Fault (exception 8) will also be generated when the processor attempts to invoke the Page Fault (exception 14) service routine, and detects an exception other than a second Page Fault. In any functional system, the entire Page Fault service routine must remain "present" in memory.

Double page faults however do not raise the double fault exception. If a second page fault occurs while the processor is attempting to enter the service routine for the first time, then the processor will invoke

the page fault (exception 14) handler a second time, rather than the double fault (exception 8) handler. A subsequent fault, though, will lead to shutdown.

When a Double Fault occurs, the 386 DX invokes the exception service routine for exception 8.

2.10 RESET AND INITIALIZATION

When the processor is initialized or Reset the registers have the values shown in Table 2-7. The 386 DX will then start executing instructions near the top of physical memory, at location FFFFFFF0H. When the first InterSegment Jump or Call is executed, address lines A20-31 will drop low for CS-relative memory cycles, and the 386 DX will only execute instructions in the lower one megabyte of physical memory. This allows the system designer to use a ROM at the top of physical memory to initialize the system and take care of Resets.

RESET forces the 386 DX to terminate all execution and local bus activity. No instruction execution or bus activity will occur as long as Reset is active. Between 350 and 450 CLK2 periods after Reset becomes inactive the 386 DX will start executing instructions at the top of physical memory.

Table 2-7. Register Values after Reset

Flag Word	UUUU0002H	Note 1
Machine Status Word (CR0)	UUUUUUU0H	Note 2
Instruction Pointer	0000FFF0H	
Code Segment	F000H	Note 3
Data Segment	0000H	
Stack Segment	0000H	
Extra Segment (ES)	0000H	
Extra Segment (FS)	0000H	
Extra Segment (GS)	0000H	
DX register	component and stepping ID	Note 5
All other registers	undefined	Note 4

NOTES:

1. EFLAG Register. The upper 14 bits of the EFLAGS register are undefined, VM (Bit 17) and RF (BIT) 16 are 0 as are all other defined flag bits.
2. CR0: (Machine Status Word). All of the defined fields in the CR0 are 0 (PG Bit 31, TS Bit 3, EM Bit 2, MP Bit 1, and PE Bit 0).
3. The Code Segment Register (CS) will have its Base Address set to FFFF0000H and Limit set to 0FFFFH.
4. All undefined bits are Intel Reserved and should not be used.
5. DX register always holds component and stepping identifier (see 5.7). EAX register holds self-test signature if self-test was requested (see 5.6).

2.11 TESTABILITY

2.11.1 Self-Test

The 386 DX has the capability to perform a self-test. The self-test checks the function of all of the Control ROM and most of the non-random logic of the part. Approximately one-half of the 386 DX can be tested during self-test.

Self-Test is initiated on the 386 DX when the RESET pin transitions from HIGH to LOW, and the BUSY# pin is low. The self-test takes about 2¹⁹ clocks, or approximately 26 milliseconds with a 20 MHz 386 DX. At the completion of self-test the processor performs reset and begins normal operation. The part has successfully passed self-test if the contents of the EAX register are zero (0). If the results of EAX are not zero then the self-test has detected a flaw in the part.

2.11.2 TLB Testing

The 386 DX provides a mechanism for testing the Translation Lookaside Buffer (TLB) if desired. This particular mechanism is unique to the 386 DX and may not be continued in the same way in future processors. When testing the TLB paging must be turned off (PG = 0 in CR0) to enable the TLB testing hardware and avoid interference with the test data being written to the TLB.

There are two TLB testing operations: 1) write entries into the TLB, and, 2) perform TLB lookups. Two Test Registers, shown in Figure 2-12, are provided for the purpose of testing. TR6 is the "test command register", and TR7 is the "test data register". The fields within these registers are defined below.

C: This is the command bit. For a write into TR6 to cause an immediate write into the TLB entry, write a 0 to this bit. For a write into TR6 to cause an immediate TLB lookup, write a 1 to this bit.

Linear Address: This is the tag field of the TLB. On a TLB write, a TLB entry is allocated to this linear address and the rest of that TLB entry is set per the value of TR7 and the value just written into TR6. On a TLB lookup, the TLB is interrogated per this value and if one and only one TLB entry matches, the rest of the fields of TR6 and TR7 are set from the matching TLB entry.

Physical Address: This is the data field of the TLB. On a write to the TLB, the TLB entry allocated to the linear address in TR6 is set to this value. On a TLB lookup, the data field (physical address) from the TLB is read out to here.

PL: On a TLB write, PL = 1 causes the REP field of TR7 to select which of four associative blocks of the TLB is to be written, but PL = 0 allows the internal pointer in the paging unit to select which TLB block is written. On a TLB lookup, the PL bit indicates whether the lookup was a hit (PL gets set to 1) or a miss (PL gets reset to 0).

V: The valid bit for this TLB entry. All valid bits can also be cleared by writing to CR3.

D, D#: The dirty bit for/from the TLB entry.

U, U#: The user bit for/from the TLB entry.

W, W#: The writable bit for/from the TLB entry.

For D, U and W, both the attribute and its complement are provided as tag bits, to permit the option of a "don't care" on TLB lookups. The meaning of these pairs of bits is given in the following table:

X	X#	Effect During TLB Lookup	Value of Bit X after TLB Write
0	0	Miss All	Bit X Becomes Undefined
0	1	Match if X = 0	Bit X Becomes 0
1	0	Match if X = 1	Bit X Becomes 1
1	1	Match all	Bit X Becomes Undefined

For writing a TLB entry:

1. Write TR7 for the desired physical address, PL and REP values.
2. Write TR6 with the appropriate linear address, etc. (be sure to write C = 0 for "write" command).

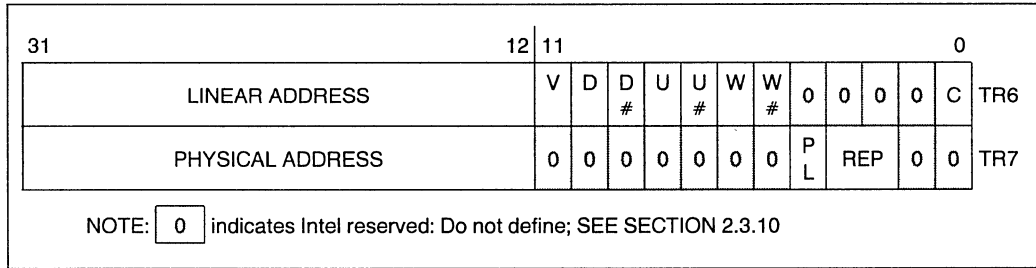
For looking up (reading) a TLB entry:

1. Write TR6 with the appropriate linear address (be sure to write C = 1 for "lookup" command).
2. Read TR7 and TR6. If the PL bit in TR7 indicates a hit, then the other values reveal the TLB contents. If PL indicates a miss, then the other values in TR7 and TR6 are indeterminate.

2.12 DEBUGGING SUPPORT

The 386 DX provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

- 1) the code execution breakpoint opcode (0CCH),
- 2) the single-step capability provided by the TF bit in the flag register, and
- 3) the code and data breakpoint capability provided by the Debug Registers DR0-3, DR6, and DR7.


Figure 2-12. Test Registers

2.12.1 Breakpoint Instruction

A single-byte-opcode breakpoint instruction is available for use by software debuggers. The breakpoint opcode is 0CCh, and generates an exception 3 trap when executed. In typical use, a debugger program can “plant” the breakpoint instruction at all desired code execution breakpoints. The single-byte breakpoint opcode is an alias for the two-byte general software interrupt instruction, INT n, where n=3. The only difference between INT 3 (0CCh) and INT n is that INT 3 is never IOPL-sensitive but INT n is IOPL-sensitive in Protected Mode and Virtual 8086 Mode.

2.12.2 Single-Step Trap

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1. Precisely, exception 1 occurs as a trap after the instruction following the instruction which set TF. In typical practice, a debugger sets the TF bit of a flag register image on the debugger's stack. It then typically transfers control to the user program and loads the flag image with a signal instruction, the IRET instruction. The single-step trap occurs after executing one instruction of the user program.

Since the exception 1 occurs as a trap (that is, it occurs after the instruction has already executed), the CS:EIP pushed onto the debugger's stack points to the next unexecuted instruction of the program being debugged. An exception 1 handler, merely by ending with an IRET instruction, can therefore efficiently support single-stepping through a user program.

2.12.3 Debug Registers

The Debug Registers are an advanced debugging feature of the 386 DX. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be

placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT3 breakpoint opcode.

The 386 DX contains six Debug Registers, providing the ability to specify up to four distinct breakpoints addresses, breakpoint control options, and read breakpoint status. Initially after reset, breakpoints are in the disabled state. Therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are autovectored to exception number 1.

2.12.3.1 LINEAR ADDRESS BREAKPOINT REGISTERS (DR0–DR3)

Up to four breakpoint addresses can be specified by writing into Debug Registers DR0–DR3, shown in Figure 2-13. The breakpoint addresses specified are 32-bit linear addresses. 386 DX hardware continuously compares the linear breakpoint addresses in DR0–DR3 with the linear addresses generated by executing software (a linear address is the result of computing the effective address and adding the 32-bit segment base address). Note that if paging is not enabled the linear address equals the physical address. If paging is enabled, the linear address is translated to a physical 32-bit address by the on-chip paging unit. Regardless of whether paging is enabled or not, however, the breakpoint registers hold linear addresses.

2.12.3.2 DEBUG CONTROL REGISTER (DR7)

A Debug Control Register, DR7 shown in Figure 2-13, allows several debug control functions such as enabling the breakpoints and setting up other control options for the breakpoints. The fields within the Debug Control Register, DR7, are as follows:

LENi (breakpoint length specification bits)

A 2-bit LEN field exists for each of the four breakpoints. LEN specifies the length of the associated breakpoint field. The choices for data breakpoints are: 1 byte, 2 bytes, and 4 bytes. Instruction execu-

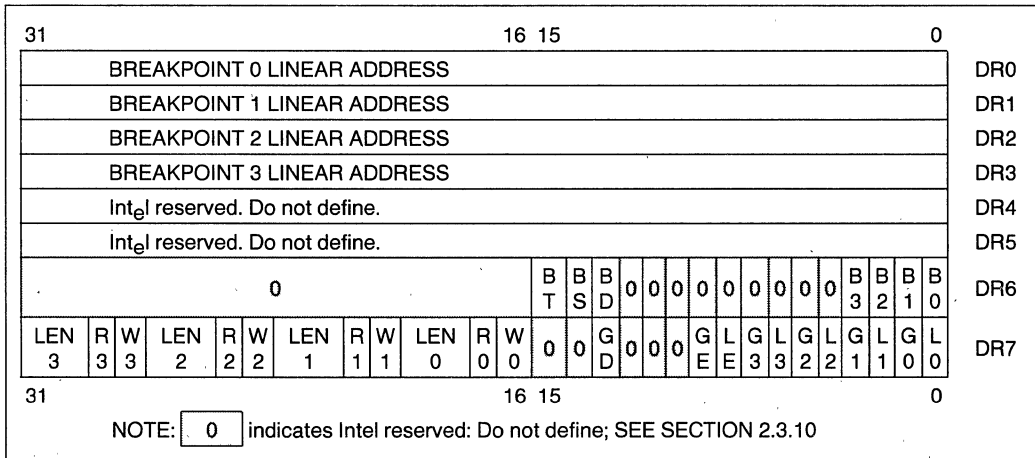


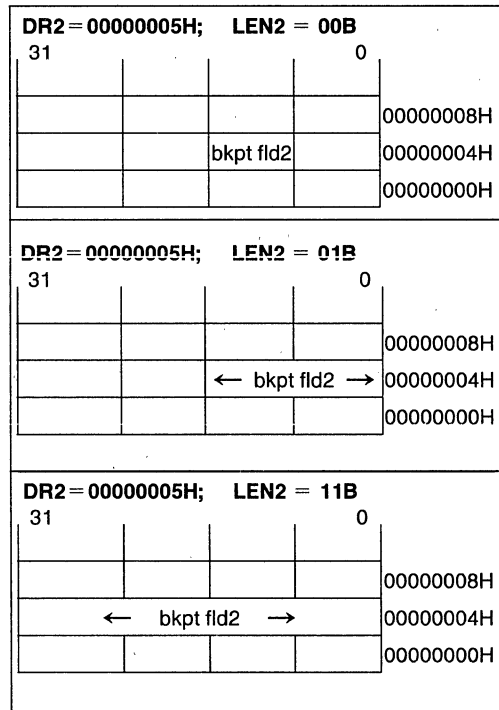
Figure 2-13. Debug Registers

tion breakpoints must have a length of 1 (LENi = 00). Encoding of the LENi field is as follows:

LENi Encoding	Breakpoint Field Width	Usage of Least Significant Bits in Breakpoint Address Register i, (i = 0 - 3)
00	1 byte	All 32-bits used to specify a single-byte breakpoint field.
01	2 bytes	A1-A31 used to specify a two-byte, word-aligned breakpoint field. A0 in Breakpoint Address Register is not used.
10	Undefined—do not use this encoding	
11	4 bytes	A2-A31 used to specify a four-byte, dword-aligned breakpoint field. A0 and A1 in Breakpoint Address Register are not used.

The LENi field controls the size of breakpoint field i by controlling whether all low-order linear address bits in the breakpoint address register are used to detect the breakpoint event. Therefore, all breakpoint fields are aligned; 2-byte breakpoint fields begin on Word boundaries, and 4-byte breakpoint fields begin on Dword boundaries.

The following is an example of various size breakpoint fields. Assume the breakpoint linear address in DR2 is 00000005H. In that situation, the following illustration indicates the region of the breakpoint field for lengths of 1, 2, or 4 bytes.



RW_i (memory access qualifier bits)

A 2-bit RW field exists for each of the four breakpoints. The 2-bit RW field specifies the type of usage which must occur in order to activate the associated breakpoint.

RW Encoding	Usage Causing Breakpoint
00	Instruction execution only
01	Data writes only
10	Undefined—do not use this encoding
11	Data reads and writes only

RW encoding 00 is used to set up an instruction execution breakpoint. RW encodings 01 or 11 are used to set up write-only or read/write data breakpoints.

Note that **instruction execution breakpoints are taken as faults** (i.e. before the instruction executes), but **data breakpoints are taken as traps** (i.e. after the data transfer takes place).

Using LEN_i and RW_i to Set Data Breakpoint *i*

A data breakpoint can be set up by writing the linear address into DR_i (*i* = 0–3). For data breakpoints, RW_i can = 01 (write-only) or 11 (write/read). LEN can = 00, 01, or 11.

If a data access entirely or partly falls within the data breakpoint field, the data breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 trap will occur.

Using LEN_i and RW_i to Set Instruction Execution Breakpoint *i*

An instruction execution breakpoint can be set up by writing address of the beginning of the instruction (including prefixes if any) into DR_i (*i* = 0–3). RW_i must = 00 and LEN must = 00 for instruction execution breakpoints.

If the instruction beginning at the breakpoint address is about to be executed, the instruction execution breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 fault will occur before the instruction is executed.

Note that an instruction execution breakpoint address must be equal to the **beginning** byte address of an instruction (including prefixes) in order for the instruction execution breakpoint to occur.

GD (Global Debug Register access detect)

The Debug Registers can only be accessed in Real Mode or at privilege level 0 in Protected Mode. The

GD bit, when set, provides extra protection against **any** Debug Register access even in Real Mode or at privilege level 0 in Protected Mode. This additional protection feature is provided to guarantee that a software debugger (or ICE™-386) can have full control over the Debug Register resources when required. The GD bit, when set, causes an exception 1 fault if an instruction attempts to read or write any Debug Register. The GD bit is then automatically cleared when the exception 1 handler is invoked, allowing the exception 1 handler free access to the debug registers.

GE and LE (Exact data breakpoint match, global and local)

If either GE or LE is set, any data breakpoint trap will be reported exactly after completion of the instruction that caused the operand transfer. Exact reporting is provided by forcing the 386 DX execution unit to wait for completion of data operand transfers before beginning execution of the next instruction.

If exact data breakpoint match is not selected, data breakpoints may not be reported until several instructions later or may not be reported at all. When enabling a data breakpoint, it is therefore recommended to enable the exact data breakpoint match.

When the 386 DX performs a task switch, the LE bit is cleared. Thus, the LE bit supports fast task switching out of tasks, that have enabled the exact data breakpoint match for their task-local breakpoints. The LE bit is cleared by the processor during a task switch, to avoid having exact data breakpoint match enabled in the new task. Note that exact data breakpoint match must be re-enabled under software control.

The 386 DX GE bit is unaffected during a task switch. The GE bit supports exact data breakpoint match that is to remain enabled during all tasks executing in the system.

Note that **instruction execution** breakpoints are always reported exactly, whether or not exact data breakpoint match is selected.

G_i and L_i (breakpoint enable, global and local)

If either G_i or L_i is set then the associated breakpoint (as defined by the linear address in DR_i, the length in LEN_i and the usage criteria in RW_i) is enabled. If either G_i or L_i is set, and the 386 DX detects the *i*th breakpoint condition, then the exception 1 handler is invoked.

When the 386 DX performs a task switch to a new Task State Segment (TSS), all L_i bits are cleared. Thus, the L_i bits support fast task switching out of tasks that use some task-local breakpoint

registers. The Li bits are cleared by the processor during a task switch, to avoid spurious exceptions in the new task. Note that the breakpoints must be re-enabled under software control.

All 386 DX Gi bits are unaffected during a task switch. The Gi bits support breakpoints that are active in all tasks executing in the system.

2.12.3.3 DEBUG STATUS REGISTER (DR6)

A Debug Status Register, DR6 shown in Figure 2-13, allows the exception 1 handler to easily determine why it was invoked. Note the exception 1 handler can be invoked as a result of one of several events:

- 1) DR0 Breakpoint fault/trap.
- 2) DR1 Breakpoint fault/trap.
- 3) DR2 Breakpoint fault/trap.
- 4) DR3 Breakpoint fault/trap.
- 5) Single-step (TF) trap.
- 6) Task switch trap.
- 7) Fault due to attempted debug register access when GD=1.

The Debug Status Register contains single-bit flags for each of the possible events invoking exception 1. Note below that some of these events are faults (exception taken before the instruction is executed), while other events are traps (exception taken after the debug events occurred).

The flags in DR6 are set by the hardware but never cleared by hardware. Exception 1 handler software should clear DR6 before returning to the user program to avoid future confusion in identifying the source of exception 1.

The fields within the Debug Status Register, DR6, are as follows:

Bi (debug fault/trap due to breakpoint 0-3)

Four breakpoint indicator flags, B0-B3, correspond one-to-one with the breakpoint registers in DR0-DR3. A flag Bi is set when the condition described by DRi, LENi, and RWi occurs.

If Gi or Li is set, and if the ith breakpoint is detected, the processor will invoke the exception 1 handler. The exception is handled as a fault if an instruction execution breakpoint occurred, or as a trap if a data breakpoint occurred.

IMPORTANT NOTE: A flag Bi is set whenever the hardware detects a match condition on **enabled** breakpoint i. Whenever a match is detected on at least one **enabled** breakpoint i, the hardware imme-

diately sets all Bi bits corresponding to breakpoint conditions matching at that instant, whether enabled **or not**. Therefore, the exception 1 handler may see that multiple Bi bits are set, but only set Bi bits corresponding to **enabled** breakpoints (Li or Gi set) are **true** indications of why the exception 1 handler was invoked.

BD (debug fault due to attempted register access when GD bit set)

This bit is set if the exception 1 handler was invoked due to an instruction attempting to read or write to the debug registers when GD bit was set. If such an event occurs, then the GD bit is automatically cleared when the exception 1 handler is invoked, allowing handler access to the debug registers.

BS (debug trap due to single-step)

This bit is set if the exception 1 handler was invoked due to the TF bit in the flag register being set (for single-stepping). See section 2.12.2.

BT (debug trap due to task switch)

This bit is set if the exception 1 handler was invoked due to a task switch occurring to a task having a 386 DX TSS with the T bit set. (See Figure 4-15a). Note the task switch into the new task occurs normally, but before the first instruction of the task is executed, the exception 1 handler is invoked. With respect to the task switch operation, the operation is considered to be a trap.

2.12.3.4 USE OF RESUME FLAG (RF) IN FLAG REGISTER

The Resume Flag (RF) in the flag word can suppress an instruction execution breakpoint when the exception 1 handler returns to a user program at a user address which is also an instruction execution breakpoint. See section 2.3.3.

3. REAL MODE ARCHITECTURE

3.1 REAL MODE INTRODUCTION

When the processor is reset or powered up it is initialized in Real Mode. Real Mode has the same base architecture as the 8086, but allows access to the 32-bit register set of the 386 DX. The addressing mechanism, memory size, interrupt handling, are all identical to the Real Mode on the 80286.

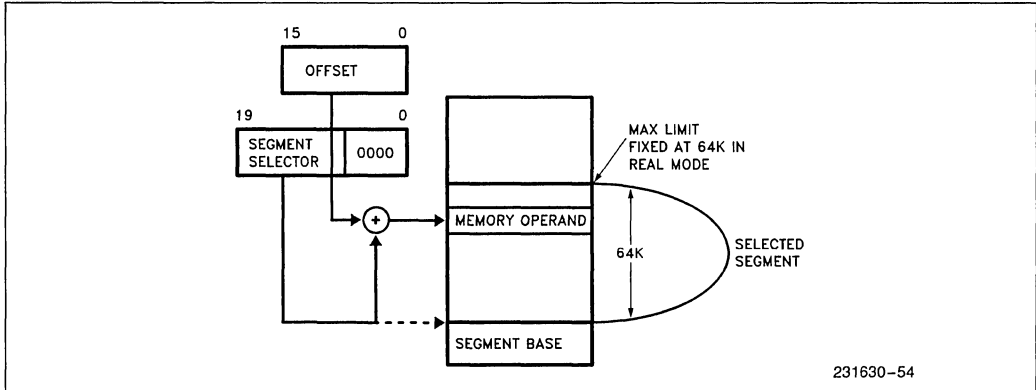


Figure 3-1. Real Address Mode Addressing

All of the 386 DX instructions are available in Real Mode (except those instructions listed in 4.6.4). The default operand size in Real Mode is 16-bits, just like the 8086. In order to use the 32-bit registers and addressing modes, override prefixes must be used. In addition, the segment size on the 386 DX in Real Mode is 64K bytes so 32-bit effective addresses must have a value less the 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode Operation.

The LOCK prefix on the 386 DX, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the 386 DX in Protected Mode and Virtual 8086 Mode. Paging makes it impossible to guarantee that repeated string instructions can be LOCKed. The 386 DX can't require that all pages holding the string be physically present in memory. Hence, a Page Fault (exception 14) might have to be taken during the repeated string instruction. Therefore the LOCK prefix can't be supported during repeated string instructions.

These are the only instruction forms where the LOCK prefix is legal on the 386 DX:

Opcode	Operands (Dest, Source)
BIT Test and SET/RESET/COMPLEMENT	Mem, Reg/immed
XCHG	Reg, Mem
XCHG	Mem, Reg
ADD, OR, ADC, SBB, AND, SUB, XOR	Mem, Reg/immed
NOT, NEG, INC, DEC	Mem

An exception 6 will be generated if a LOCK prefix is placed before any instruction form or opcode not listed above. The LOCK prefix allows indivisible

read/modify/write operations on memory operands using the instructions above. For example, even the ADD Reg, Mem is not LOCKable, because the Mem operand is not the destination (and therefore no memory read/modify/operation is being performed).

Since, on the 386 DX, repeated string instructions are not LOCKable, it is not possible to LOCK the bus for a long period of time. Therefore, the LOCK prefix is not IOPL-sensitive on the 386 DX. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed above.

3.2 MEMORY ADDRESSING

In Real Mode the maximum memory size is limited to 1 megabyte. Thus, only address lines A2–A19 are active. (Exception, the high address lines A20–A31 are high during CS-relative memory cycles until an intersegment jump or call is executed (see section 2.10)).

Since paging is not allowed in Real Mode the linear addresses are the same as physical addresses. Physical addresses are formed in Real Mode by adding the contents of the appropriate segment register which is shifted left by four bits to an effective address. This addition results in a physical address from 00000000H to 0010FFFFH. This is compatible with 80286 Real Mode. Since segment registers are shifted left by 4 bits this implies that Real Mode segments always start on 16 byte boundaries.

All segments in Real Mode are exactly 64K bytes long, and may be read, written, or executed. The 386 DX will generate an exception 13 if a data operand or instruction fetch occurs past the end of a segment. (i.e. if an operand has an offset greater than FFFFH, for example a word with a low byte at FFFFH and the high byte at 0000H.)

Segments may be overlapped in Real Mode. Thus, if a particular segment does not use all 64K bytes another segment can be overlaid on top of the unused portion of the previous segment. This allows the programmer to minimize the amount of physical memory needed for a program.

3.3 RESERVED LOCATIONS

There are two fixed areas in memory which are reserved in Real address mode: system initialization area and the interrupt table area. Locations 00000H through 003FFFH are reserved for interrupt vectors. Each one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations FFFFFFF0H through FFFFFFFFH are reserved for system initialization.

3.4 INTERRUPTS

Many of the exceptions shown in Table 2-5 and discussed in section 2.9 are not applicable to Real Mode operation, in particular exceptions 10, 11, 14, will not happen in Real Mode. Other exceptions have slightly different meanings in Real Mode; Table 3-1 identifies these exceptions.

3.5 SHUTDOWN AND HALT

The HLT instruction stops program execution and prevents the processor from using the local bus until restarted. Either NMI, INTR with interrupts enabled (IF = 1), or RESET will force the 386 DX out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

Shutdown will occur when a severe error is detected that prevents further processing. In Real Mode, shutdown can occur under two conditions:

An interrupt or an exception occur (Exceptions 8 or 13) and the interrupt vector is larger than the

Interrupt Descriptor Table (i.e. There is not an interrupt handler for the interrupt).

A CALL, INT or PUSH instruction attempts to wrap around the stack segment when SP is not even. (e.g. pushing a value on the stack when SP = 0001 resulting in a stack segment greater than FFFFH)

An NMI input can bring the processor out of shutdown if the Interrupt Descriptor Table limit is large enough to contain the NMI interrupt vector (at least 0017H) and the stack has enough room to contain the vector and flag information (i.e. SP is greater than 0005H). Otherwise shutdown can only be exited via the RESET input.

4. PROTECTED MODE ARCHITECTURE

4.1 INTRODUCTION

The complete capabilities of the 386 DX are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to four gigabytes (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 terabytes or 2^{46} bytes). In addition Protected Mode allows the 386 DX to run all of the existing 8086 and 80286 software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions especially optimized for supporting multitasking operating systems. The base architecture of the 386 DX remains the same, the registers, instructions, and addressing modes described in the previous sections are retained. The main difference between Protected Mode, and Real Mode from a programmer's view is the increased address space, and a different addressing mechanism.

Table 3-1

Function	Interrupt Number	Related Instructions	Return Address Location
Interrupt table limit too small	8	INT Vector is not within table limit	Before Instruction
CS, DS, ES, FS, GS Segment overrun exception	13	Word memory reference beyond offset = FFFFH. An attempt to execute past the end of CS segment.	Before Instruction
SS Segment overrun exception	12	Stack Reference beyond offset = FFFFH	Before Instruction

4.2 ADDRESSING MECHANISM

Like Real Mode, Protected Mode uses two components to form the logical address, a 16-bit selector is used to determine the linear base address of a segment, the base address is added to a 32-bit effective address to form a 32-bit linear address. The linear address is then either used as the 32-bit physical address, or if paging is enabled the paging mechanism maps the 32-bit linear address into a 32-bit physical address.

The difference between the two modes lies in calculating the base address. In Protected Mode the selector is used to specify an index into an operating

system defined table (see Figure 4-1). The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base address obtained from the table to the offset.

Paging provides an additional memory management mechanism which operates only in Protected Mode. Paging provides a means of managing the very large segments of the 386 DX. As such, paging operates beneath segmentation. The paging mechanism translates the protected linear address which comes from the segmentation unit into a physical address. Figure 4-2 shows the complete 386 DX addressing mechanism with paging enabled.

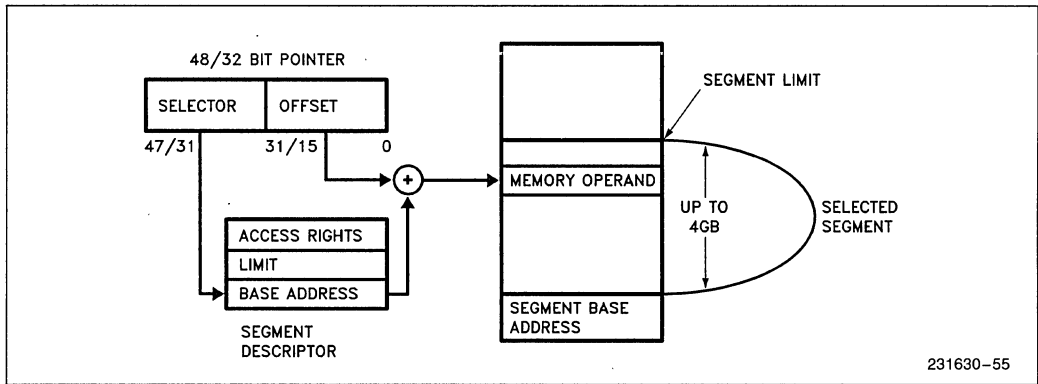


Figure 4-1. Protected Mode Addressing

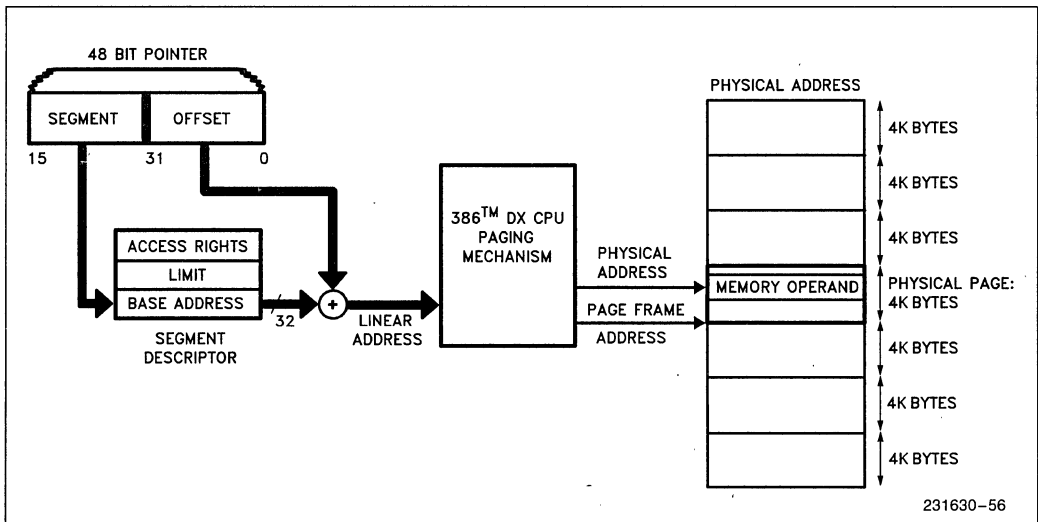


Figure 4-2. Paging and Segmentation

4.3 SEGMENTATION

4.3.1 Segmentation Introduction

Segmentation is one method of memory management. Segmentation provides the basis for protection. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about a segment is stored in an 8 byte data structure called a descriptor. All of the descriptors in a system are contained in tables recognized by hardware.

4.3.2 Terminology

The following terms are used throughout the discussion of descriptors, privilege levels and protection:

PL: Privilege Level—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged. More privileged levels are numerically smaller than less privileged levels.

RPL: Requestor Privilege Level—The privilege level of the original supplier of the selector. RPL is determined by the **least two** significant bits of a selector.

DPL: Descriptor Privilege Level—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.

CPL: Current Privilege Level—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.

EPL: Effective Privilege Level—The effective privilege level is the least privileged of the RPL and DPL. Since smaller privilege level **values** indicate greater privilege, EPL is the numerical maximum of RPL and DPL.

Task: One instance of the execution of a program. Tasks are also referred to as processes.

4.3.3 Descriptor Tables

4.3.3.1 DESCRIPTOR TABLES INTRODUCTION

The descriptor tables define all of the segments which are used in an 386 DX system. There are three types of tables on the 386 DX which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays. They can range in size between 8 bytes and 64K bytes. Each table can hold up to 8192 8 byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address, and the 16-bit limit of each table.

Each of the tables has a register associated with it the GDTR, LDTR, and the IDTR (see Figure 4-3). The LGDT, LLDT, and LIDT instructions, load the base and limit of the Global, Local, and Interrupt Descriptor Tables, respectively, into the appropriate register. The SGDT, SLDT, and SIDT instructions store the base and limit values. These tables are manipulated by the operating system. Therefore, the load descriptor table instructions are privileged instructions.

4.3.3.2 GLOBAL DESCRIPTOR TABLE

The Global Descriptor Table (GDT) contains descriptors which are possibly available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for descriptors which are used for servicing interrupts (i.e. interrupt and trap descriptors). Every 386 DX system contains a

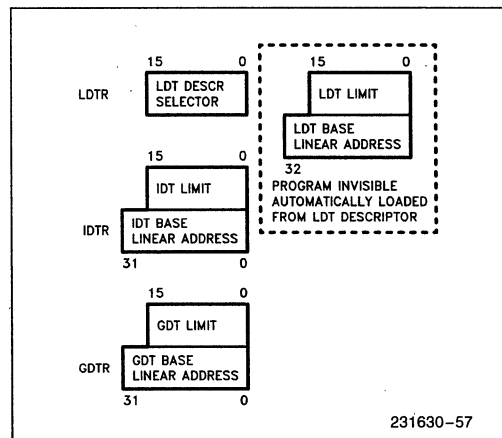


Figure 4-3. Descriptor Table Registers

GDT. Generally the GDT contains code and data segments used by the operating systems and task state segments, and descriptors for the LDTs in a system.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

4.3.3.3 LOCAL DESCRIPTOR TABLE

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This provides both isolation and protection for a task's segments, while still allowing global data to be shared among tasks.

Unlike the 6 byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT.

4.3.3.4 INTERRUPT DESCRIPTOR TABLE

The third table needed for 386 DX systems is the Interrupt Descriptor Table. (See Figure 4-4.) The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT

may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced via INT instructions, external interrupt vectors, and exceptions. (See 2.9 **Interrupts**).

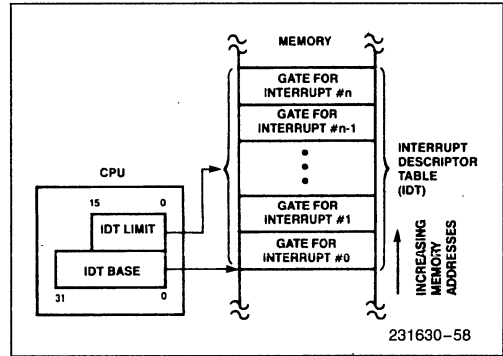


Figure 4-4. Interrupt Descriptor Table Register Use

4.3.4 Descriptors

4.3.4.1 DESCRIPTOR ATTRIBUTE BITS

The object to which the segment selector points to is called a descriptor. Descriptors are eight byte quantities which contain attributes about a given region of linear address space (i.e. a segment). These attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or

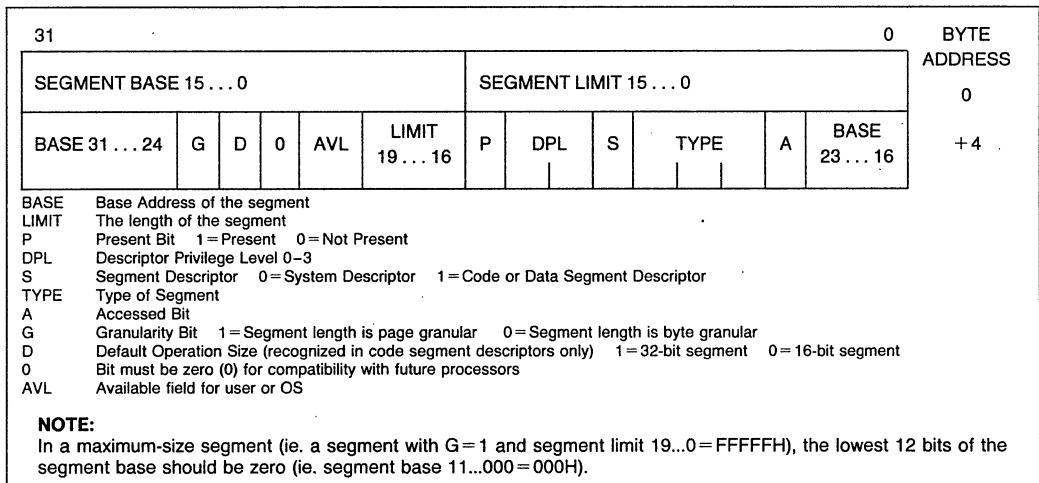


Figure 4-5. Segment Descriptors

32-bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 4-5 shows the general format of a descriptor. All segments on the 386 DX have three attribute fields in common: the **P** bit, the **DPL** bit, and the **S** bit. The Present **P** bit is 1 if the segment is loaded in physical memory, if $P=0$ then any attempt to access this segment causes a not present exception (exception 11). The Descriptor Privilege Level **DPL** is a two-bit field which specifies the protection level 0–3 associated with a segment.

The 386 DX has two main categories of segments system segments and non-system segments (for

code and data). The segment **S** bit in the segment descriptor determines if a given segment is a system segment or a code or data segment. If the **S** bit is 1 then the segment is either a code or data segment, if it is 0 then the segment is a system segment.

4.3.4.2 386™ DX CODE, DATA DESCRIPTORS (S = 1)

Figure 4-6 shows the general format of a code and data descriptor and Table 4-1 illustrates how the bits in the Access Rights Byte are interpreted.

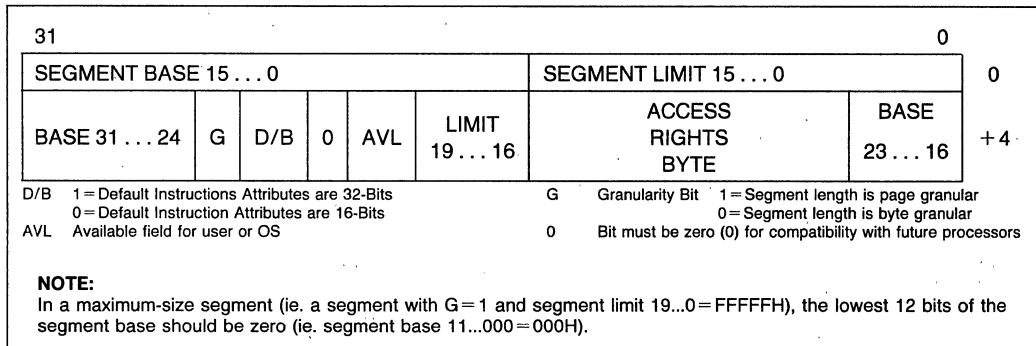


Figure 4-6. Segment Descriptors

Table 4-1. Access Rights Byte Definition for Code and Data Descriptions

Bit Position	Name	Function
7	Present (P)	$P = 1$ Segment is mapped into physical memory. $P = 0$ No mapping to physical memory exists, base and limit are not used.
6–5	Descriptor Privilege Level (DPL)	Segment privilege attribute used in privilege tests.
4	Segment Descriptor (S)	$S = 1$ Code or Data (includes stacks) segment descriptor $S = 0$ System Segment Descriptor or Gate Descriptor
3	Executable (E)	$E = 0$ Descriptor type is data segment: $ED = 0$ Expand up segment, offsets must be \leq limit. $ED = 1$ Expand down segment, offsets must be $>$ limit. $W = 0$ Data segment may not be written into. $W = 1$ Data segment may be written into.
2	Expansion Direction (ED)	
1	Writeable (W)	
3	Executable (E)	$E = 1$ Descriptor type is code segment: $C = 1$ Code segment may only be executed when $CPL \geq DPL$ and CPL remains unchanged.
2	Conforming (C)	
1	Readable (R)	$R = 0$ Code segment may not be read. $R = 1$ Code segment may be read.
0	Accessed (A)	$A = 0$ Segment has not been accessed. $A = 1$ Segment selector has been loaded into segment register or used by selector test instructions.

Type Field Definition

}	If Data Segment ($S = 1, E = 0$)
}	If Code Segment ($S = 1, E = 1$)

Code and data segments have several descriptor fields in common. The accessed **A** bit is set whenever the processor accesses a descriptor. The **A** bit is used by operating systems to keep usage statistics on a given segment. The **G** bit, or granularity bit, specifies if a segment length is byte-granular or page-granular. 386 DX segments can be one mega-byte long with byte granularity ($G=0$) or four gigabytes with page granularity ($G=1$), (i.e., 2^{20} pages each page is 4K bytes in length). The granularity is totally unrelated to paging. A 386 DX system can consist of segments with byte granularity, and page granularity, whether or not paging is enabled.

The executable **E** bit tells if a segment is a code or data segment. A code segment ($E=1, S=1$) may be execute-only or execute/read as determined by the Read **R** bit. Code segments are execute only if $R=0$, and execute/read if $R=1$. Code segments may never be written into.

NOTE:

Code segments may be modified via aliases. Aliases are writeable data segments which occupy the same range of linear address space as the code segment.

The **D** bit indicates the default length for operands and effective addresses. If $D=1$ then 32-bit operands and 32-bit addressing modes are assumed. If $D=0$ then 16-bit operands and 16-bit addressing modes are assumed. Therefore all existing 80286 code segments will execute on the 386 DX assuming the **D** bit is set 0.

Another attribute of code segments is determined by the conforming **C** bit. Conforming segments, $C=1$, can be executed and shared by programs at different privilege levels. (See section 4.4 **Protection**.)

Segments identified as data segments ($E=0, S=1$) are used for two types of 386 DX segments: stack and data segments. The expansion direction (**ED**) bit specifies if a segment expands downward (stack) or upward (data). If a segment is a stack segment all offsets must be greater than the segment limit. On a data segment all offsets must be less than or equal to the limit. In other words, stack segments start at the base linear address plus the maximum segment limit and grow down to the base linear address plus the limit. On the other hand, data segments start at the base linear address and expand to the base linear address plus limit.

The write **W** bit controls the ability to write into a segment. Data segments are read-only if $W=0$. The stack segment must have $W=1$.

The **B** bit controls the size of the stack pointer register. If $B=1$, then PUSHes, POPs, and CALLs all use the 32-bit ESP register for stack references and assume an upper limit of FFFFFFFFH. If $B=0$, stack instructions all use the 16-bit SP register and assume an upper limit of FFFFH.

4.3.4.3 SYSTEM DESCRIPTOR FORMATS

System segments describe information about operating system tables, tasks, and gates. Figure 4-7 shows the general format of system segment descriptors, and the various types of system segments. 386 DX system descriptors contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.

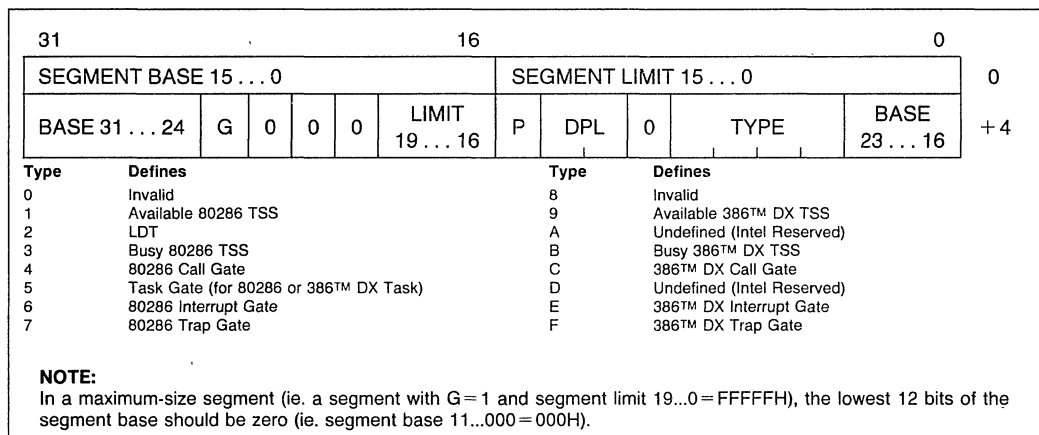


Figure 4-7. System Segments Descriptors

4.3.4.4 LDT DESCRIPTORS (S = 0, TYPE = 2)

LDT descriptors (S=0 TYPE=2) contain information about Local Descriptor Tables. LDTs contain a table of segment descriptors, unique to a particular task. Since the instruction to load the LDTR is only available at privilege level 0, the DPL field is ignored. LDT descriptors are only allowed in the Global Descriptor Table (GDT).

4.3.4.5 TSS DESCRIPTORS (S = 0, TYPE = 1, 3, 9, B)

A Task State Segment (TSS) descriptor contains information about the location, size, and privilege level of a Task State Segment (TSS). A TSS in turn is a special fixed format segment which contains all the state information for a task and a linkage field to permit nesting tasks. The TYPE field is used to indicate whether the task is currently BUSY (i.e. on a chain of active tasks) or the TSS is available. The TYPE field also indicates if the segment contains a 80286 or a 386 DX TSS. The Task Register (TR) contains the selector which points to the current Task State Segment.

4.3.4.6 GATE DESCRIPTORS (S = 0, TYPE = 4-7, C, F)

Gates are used to control access to entry points within the target code segment. The various types of

gate descriptors are **call gates**, **task gates**, **interrupt gates**, and **trap gates**. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the processor to automatically perform protection checks. It also allows system designers to control entry points to the operating system. Call gates are used to change privilege levels (see section 4.4 **Protection**), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines.

Figure 4-8 shows the format of the four types of gate descriptors. Call gates are primarily used to transfer program control to a more privileged level. The call gate descriptor consists of three fields: the access byte, a long pointer (selector and offset) which points to the start of a routine and a word count which specifies how many parameters are to be copied from the caller's stack to the stack of the called routine. The word count field is only used by call gates when there is a change in the privilege level, other types of gates ignore the word count field.

Interrupt and trap gates use the destination selector and destination offset fields of the gate descriptor as a pointer to the start of the interrupt or trap handler routines. The difference between interrupt gates and trap gates is that the interrupt gate disables interrupts (resets the IF bit) while the trap gate does not.

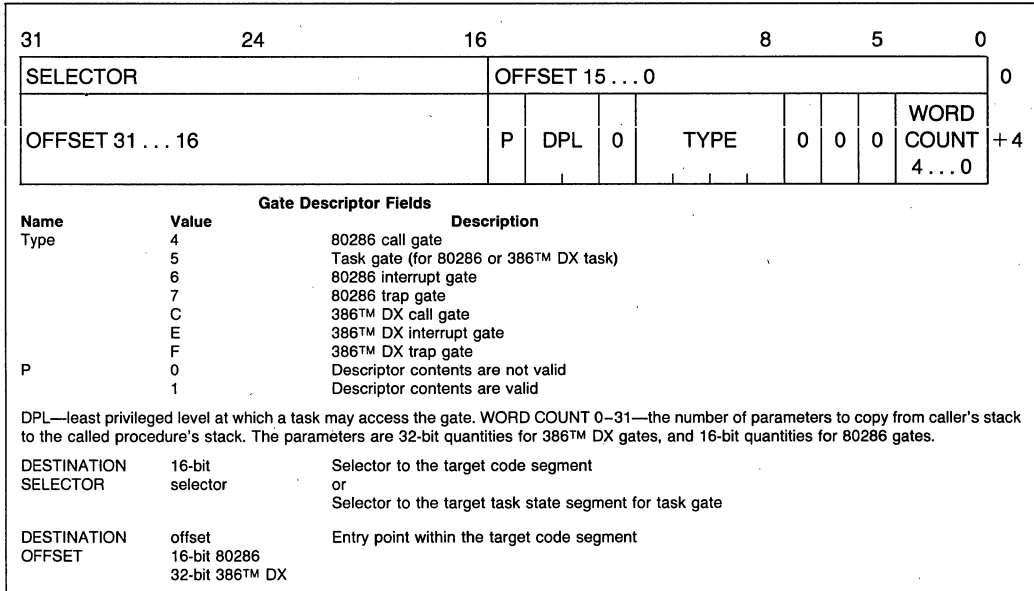


Figure 4-8. Gate Descriptor Formats

Task gates are used to switch tasks. Task gates may only refer to a task state segment (see section 4.4.6 **Task Switching**) therefore only the destination selector portion of a task gate descriptor is used, and the destination offset is ignored.

Exception 13 is generated when a destination selector does not refer to a correct descriptor type, i.e., a code segment for an interrupt, trap or call gate, a TSS for a task gate.

The access byte format is the same for all gate descriptors. P=1 indicates that the gate contents are valid. P=0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (see section 4.4 **Protection**). The S field, bit 4 of the access rights byte, must be 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 4-8.

4.3.4.7 DIFFERENCES BETWEEN 386™ DX AND 80286 DESCRIPTORS

In order to provide operating system compatibility between the 80286 and 386 DX, the 386 DX supports all of the 80286 segment descriptors. Figure 4-9 shows the general format of an 80286 system segment descriptor. The only differences between 80286 and 386 DX descriptor formats are that the values of the type fields, and the limit and base address fields have been expanded for the 386 DX. The 80286 system segment descriptors contained a 24-bit base address and 16-bit limit, while the 386 DX system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit.

By supporting 80286 system segments the 386 DX is able to execute 80286 application programs on a 386 DX operating system. This is possible because the processor automatically understands which descriptors are 80286-style descriptors and which de-

scriptors are 386 DX-style descriptors. In particular, if the upper word of a descriptor is zero, then that descriptor is a 80286-style descriptor.

The only other differences between 80286-style descriptors and 386 DX descriptors is the interpretation of the word count field of call gates and the B bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for 386 DX call gates. The B bit controls the size of PUSHes when using a call gate; if B=0 PUSHes are 16 bits, if B=1 PUSHes are 32 bits.

4.3.4.8 SELECTOR FIELDS

A selector in Protected Mode has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 4-10. The TI bits select one of two memory-based tables of descriptors (the Global Descriptor Table or the Local Descriptor Table). The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

4.3.4.9 SEGMENT DESCRIPTOR CACHE

In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Since descriptor caches only change when a segment register is changed, programs which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

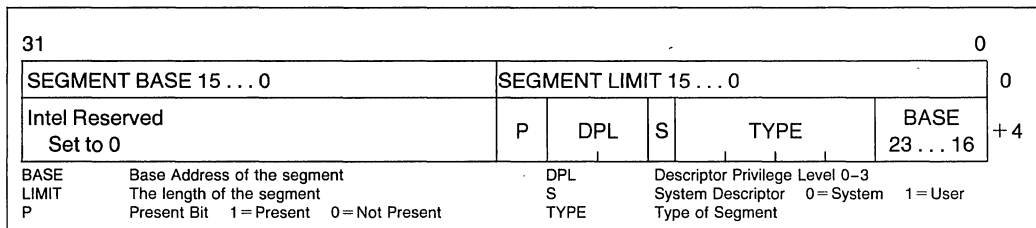


Figure 4-9. 80286 Code and Data Segment Descriptors

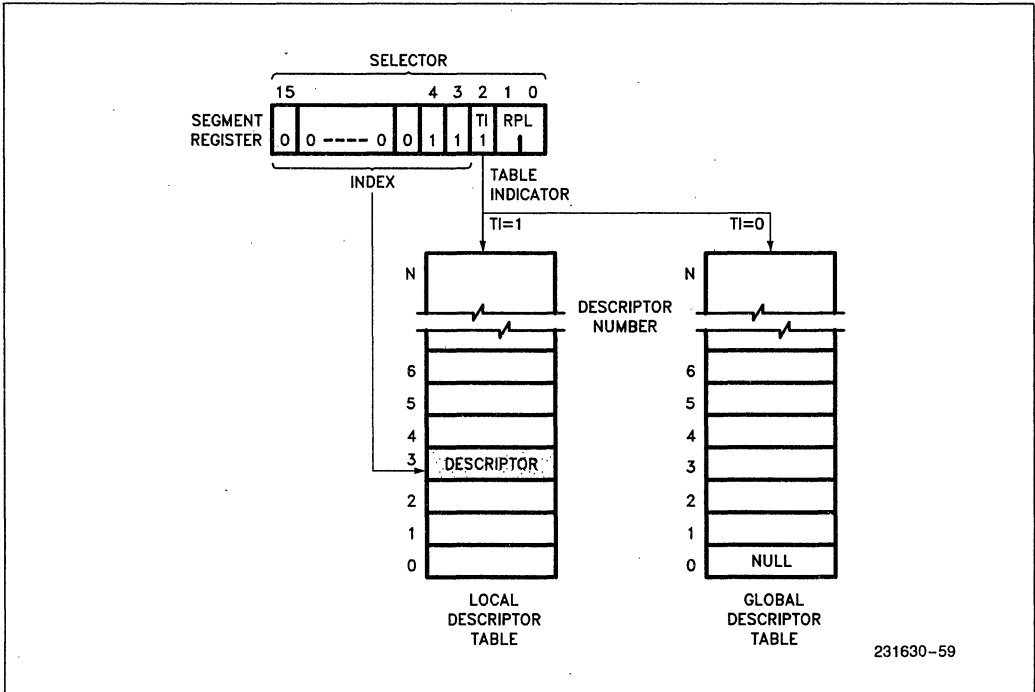


Figure 4-10. Example Descriptor Selection

231630-59

4.3.4.10 SEGMENT DESCRIPTOR REGISTER SETTINGS

The contents of the segment descriptor cache vary depending on the mode the 386 DX is operating in. When operating in Real Address Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4-11.

For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. In Real Address Mode, the internal "privilege level" is always fixed to the highest level, level 0, so I/O and other privileged opcodes may be executed.

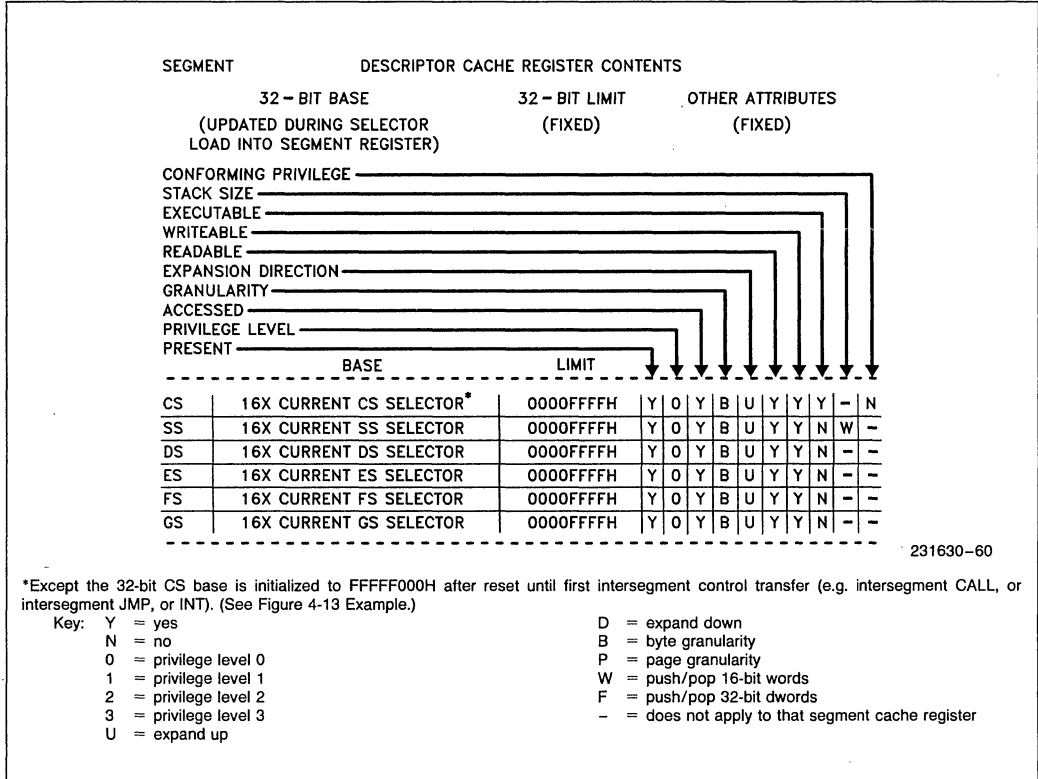


Figure 4-11. Segment Descriptor Caches for Real Address Mode (Segment Limit and Attributes are Fixed)

When operating in Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4-12. In Protected Mode, each of these fields are defined

according to the contents of the segment descriptor indexed by the selector value loaded into the segment register.

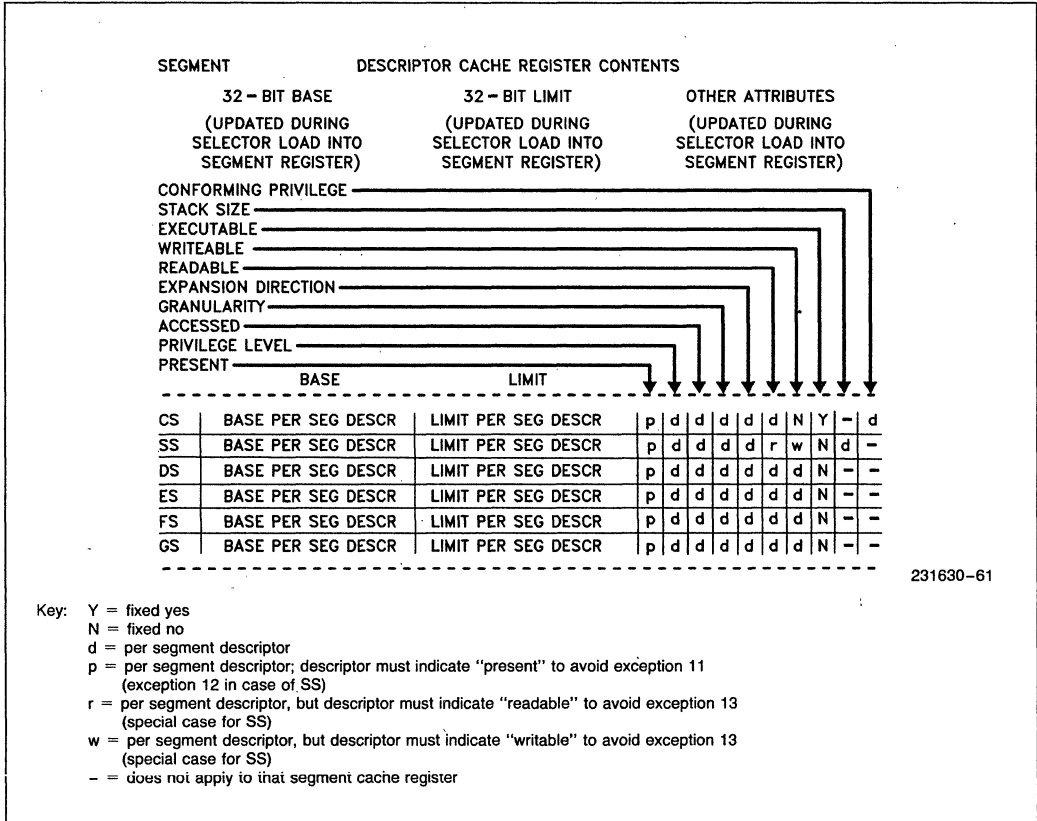


Figure 4-12. Segment Descriptor Caches for Protected Mode (Loaded per Descriptor)

When operating in a Virtual 8086 Mode within the Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4-13. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at

0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. The virtual program executes at lowest privilege level, level 3, to allow trapping of all IOPL-sensitive instructions and level-0-only instructions.

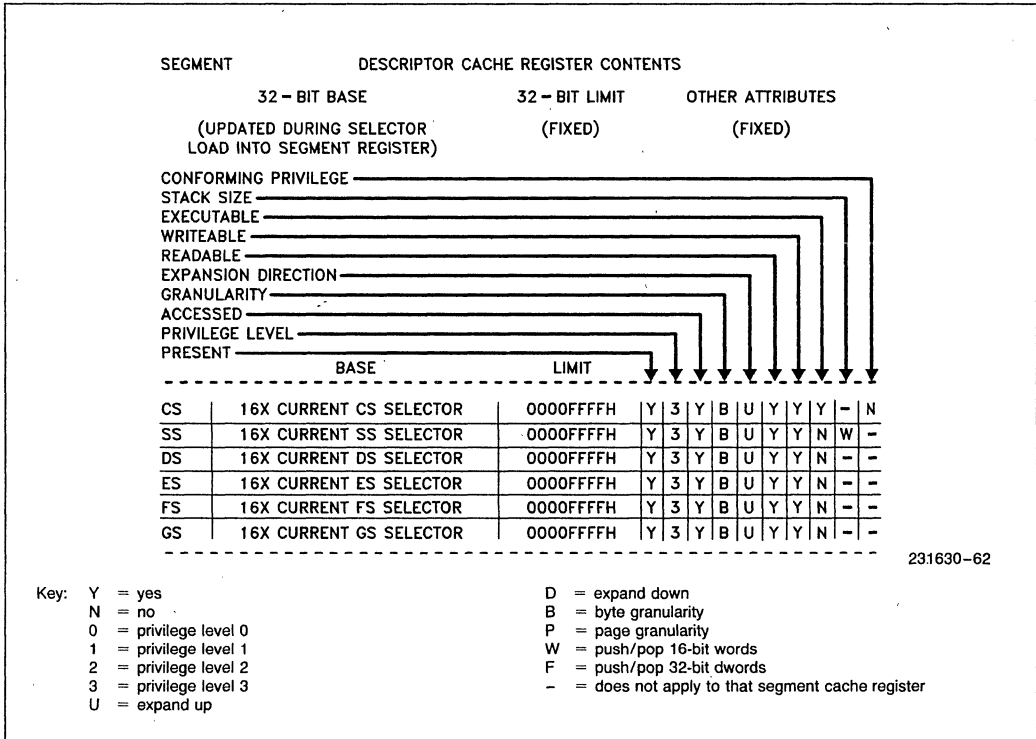


Figure 4-13. Segment Descriptor Caches for Virtual 8086 Mode within Protected Mode (Segment Limit and Attributes are Fixed)

4.4 PROTECTION

4.4.1 Protection Concepts

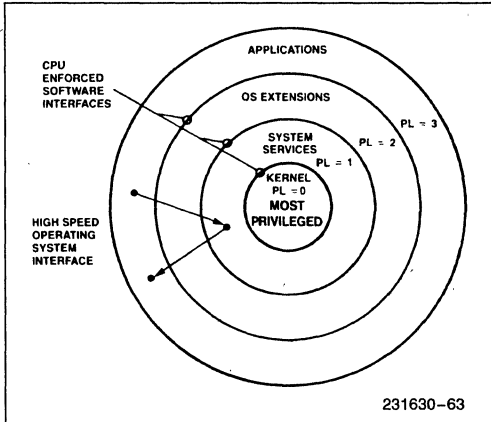


Figure 4-14. Four-Level Hierarchical Protection

The 386 DX has four levels of protection which are optimized to support the needs of a multi-tasking operating system to isolate and protect user programs from each other and the operating system. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. Unlike traditional microprocessor-based systems where this protection is achieved only through the use of complex external hardware and software the 386 DX provides the protection as part of its integrated Memory Management Unit. The 386 DX offers an additional type of protection on a page basis, when paging is enabled (See section 4.5.3 Page Level Protection).

The four-level hierarchical privilege system is illustrated in Figure 4-14. It is an extension of the user/supervisor privilege mode commonly used by mini-computers and, in fact, the user/supervisor mode is fully supported by the 386 DX paging mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged or trusted level.

4.4.2 Rules of Privilege

The 386 DX controls access to both data and procedures between levels of a task, according to the following rules.

- Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p**.
- A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p**.

4.4.3 Privilege Levels

4.4.3.1 TASK PRIVILEGE

At any point in time, a task on the 386 DX always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies the task's privilege level. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. (See section 4.4.4 Privilege Level Transfers) Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

4.4.3.2 SELECTOR PRIVILEGE (RPL)

The privilege level of a selector is specified by the RPL field. The RPL is the two least significant bits of the selector. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (i.e. numerically larger) level of a task's CPL and a selector's RPL. Thus, if selector's RPL = 0 then the CPL always specifies the privilege level for making an access using the selector. On the other hand if RPL = 3 then a selector can only access segments at level 3 regardless of the task's CPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

4.4.3.3 I/O PRIVILEGE AND I/O PERMISSION BITMAP

The I/O privilege level (IOPL, a 2-bit field in the EFLAG register) defines the least privileged level at which I/O instructions can be unconditionally performed. I/O instructions can be unconditionally performed when $CPL \leq IOPL$. (The I/O instructions are IN, OUT, INS, OUTS, REP INS, and REP OUTS.) When $CPL > IOPL$, and the current task is associated with a 286 TSS, attempted I/O instructions cause an exception 13 fault. When $CPL > IOPL$, and the current task is associated with a 386 DX TSS, the I/O Permission Bitmap (part of a 386 DX TSS) is consulted on whether I/O to the port is allowed, or an exception 13 fault is to be generated instead. For

diagrams of the I/O Permission Bitmap, refer to Figures 4-15a and 4-15b. For further information on how the I/O Permission Bitmap is used in Protected Mode or in Virtual 8086 Mode, refer to section 4.6.4 Protection and I/O Permission Bitmap.

The I/O privilege level (IOPL) also affects whether several other instructions can be executed or cause an exception 13 fault instead. These instructions are called "IOPL-sensitive" instructions and they are CLI and STI. (Note that the LOCK prefix is *not* IOPL-sensitive on the 386 DX.)

The IOPL also affects whether the IF (interrupts enable flag) bit can be changed by loading a value into the EFLAGS register. When $CPL \leq IOPL$, then the IF bit can be changed by loading a new value into the EFLAGS register. When $CPL > IOPL$, the IF bit cannot be changed by a new value POP'ed into (or otherwise loaded into) the EFLAGS register; the IF bit merely remains unchanged and no exception is generated.

Table 4-2. Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

4.4.3.4 PRIVILEGE VALIDATION

The 386 DX provides several instructions to speed pointer testing and help maintain system integrity by verifying that the selector value refers to an appropriate segment. Table 4-2 summarizes the selector validation procedures available for the 386 DX.

This pointer verification prevents the common problem of an application at $PL = 3$ calling a operating systems routine at $PL = 0$ and passing the operating system routine a "bad" pointer which corrupts a data structure belonging to the operating system. If the operating system routine uses the ARPL instruction to ensure that the RPL of the selector has no greater privilege than that of the caller, then this problem can be avoided.

4.4.3.5 DESCRIPTOR ACCESS

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL as described above.

Any time an instruction loads data segment registers (DS, ES, FS, GS) the 386 DX makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segments or readable code segments. The data access rules are specified in section 4.2.2 **Rules of Privilege**. The only exception to those rules is readable conforming code segments which can be accessed at any privilege level.

Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL an exception 13 (general protection/fault) is generated.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data segments. The DPL and RPL must equal the CPL. All other descriptor types or a privilege level violation will cause exception 13. A stack not present fault causes exception 12. Note that an exception 11 is used for a not-present code or data segment.

4.4.4 Privilege Level Transfers

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call

Table 4-3. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag register) = 0

**NT (Nested Task bit of flag register) = 1

or a jump to another routine. There are five types of control transfers which are summarized in Table 4-3. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only via control transfers, by using gates, task switches, and interrupt or trap gates.

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13 (e.g. JMP through a call gate, or IRET from a normal subroutine call).

In order to provide further system security, all control transfers are also subject to the privilege rules.

The privilege rules require that:

- Privilege level transitions can only occur via gates.
- JMPs can be made to a non-conforming code segment with the same privilege or to a conforming code segment with greater or equal privilege.
- CALLs can be made to a non-conforming code segment with the same privilege or via a gate to a more privileged level.
- Interrupts handled within the task obey the same privilege rules as CALLs.
- Conforming Code segments are accessible by privilege levels which are the same or less privileged than the conforming-code segment's DPL.
- Both the requested privilege level (RPL) in the selector pointing to the gate and the task's CPL

must be of equal or greater privilege than the gate's DPL.

- The code segment selected in the gate must be the same or more privileged than the task's CPL.
- Return instructions that do not switch tasks can only return control to a code segment with same or less privilege.
- Task switches can be performed by a CALL, JMP, or INT which references either a task gate or task state segment who's DPL is less privileged or the same privilege as the old task's CPL.

Any control transfer that changes CPL within a task causes a change of stacks as a result of the privilege level change. The initial values of SS:ESP for privilege levels 0, 1, and 2 are retained in the task state segment (see section 4.4.6 **Task Switching**). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and ESP registers and the previous stack pointer is pushed onto the new stack.

When RETURNing to the original privilege level, use of the lower-privileged stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words (as specified in the gate's word count field) are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

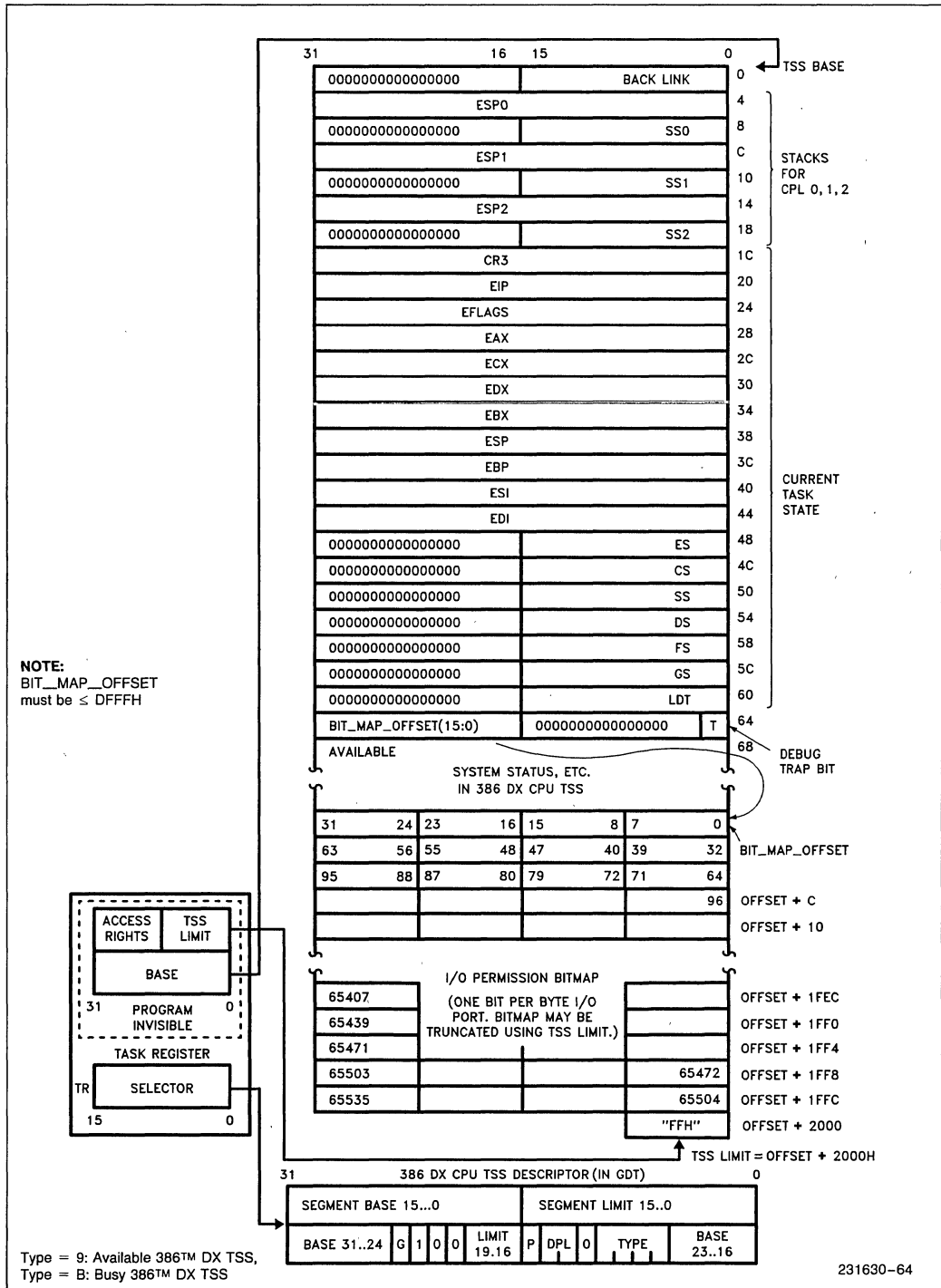


Figure 4-15a. 386™ DX TSS and TSS Registers

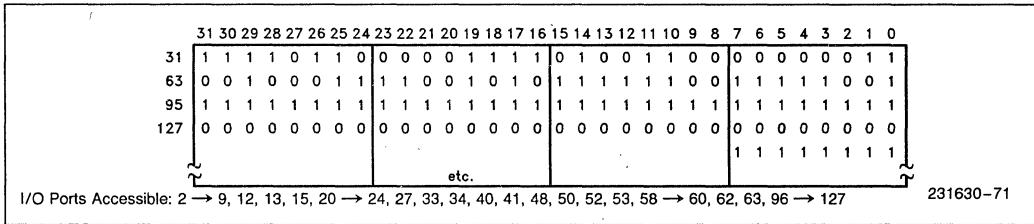


Figure 4-15b. Sample I/O Permission Bit Map

4.4.5 Call Gates

Gates provide protected, indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures (such as those which allocate memory, or perform I/O).

Gate descriptors follow the data access rules of privilege; that is, gates can be accessed by a task if the EPL, is equal to or more privileged than the gate descriptor's DPL. Gates follow the control transfer rules of privilege and therefore may only transfer control to a more privileged level.

Call Gates are accessed via a CALL instruction and are syntactically identical to calling a normal subroutine. When an inter-level 386 DX call gate is activated, the following actions occur.

1. Load CS:EIP from gate check for validity
2. SS is pushed zero-extended to 32 bits
3. ESP is pushed
4. Copy Word Count 32-bit parameters from the old stack to the new stack
5. Push Return address on stack

The procedure is identical for 80286 Call gates, except that 16-bit parameters are copied and 16-bit registers are pushed.

Interrupt Gates and Trap gates work in a similar fashion as the call gates, except there is no copying of parameters. The only difference between Trap and Interrupt gates is that control transfers through an Interrupt gate disable further interrupts (i.e. the IF bit is set to 0), and Trap gates leave the interrupt status unchanged.

4.4.6 Task Switching

A very important attribute of any multi-tasking/multi-user operating systems is its ability to rapidly switch between tasks or processes. The 386 DX directly supports this operation by providing a task switch instruction in hardware. The 386 DX task switch op-

eration saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task, in about 17 microseconds. Like transfer of control via gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 4-15) containing the entire 386 DX execution state while a task gate descriptor contains a TSS selector. The 386 DX supports both 80286 and 386 DX style TSSs. Figure 4-16 shows a 80286 TSS. The limit of a 386 DX TSS must be greater than 0064H (002BH for a 80286 TSS), and can be as large as 4 Gigabytes. In the additional TSS space, the operating system is free to store additional information such as the reason the task is inactive, time the task has spent running, and open files belong to the task.

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 386 DX called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The current executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CR0) give information about the state of a task which are useful to the operating system. The Nested Task (NT) (bit 14 in EFLAGS) controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular return; when NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

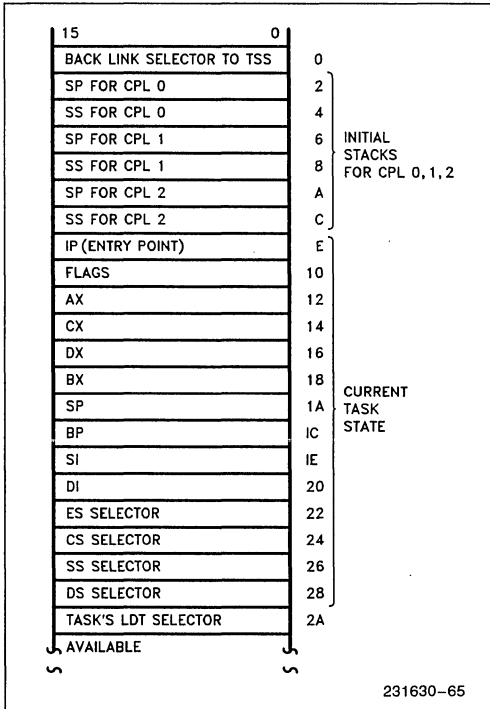


Figure 4-16. 80286 TSS

When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. (The NT bit will be restored after execution of the interrupt handler) NT may also be set or cleared by POPF or IRET instructions.

The 386 DX task state segment is marked busy by changing the descriptor type field from TYPE 9H to TYPE BH. An 80286 TSS is marked busy by changing the descriptor type field from TYPE 1 to TYPE 3. Use of a selector that references a busy task state segment causes an exception 13.

The Virtual Mode (VM) bit 17 is used to indicate if a task, is a virtual 8086 task. If VM = 1, then the tasks will use the Real Mode addressing mechanism. The virtual 8086 environment is only entered and exited via a task switch (see section 4.6 **Virtual Mode**).

The coprocessor's state is not automatically saved when a task switch occurs, because the incoming task may not use the coprocessor. The Task Switched (TS) Bit (bit 3 in the CRO) helps deal with the coprocessor's state in a multi-tasking environ-

ment. Whenever the 386 DX switches tasks, it sets the TS bit. The 386 DX detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the coprocessor. A processor extension not present exception (7) will occur when attempting to execute an ESC or WAIT instruction if the Task Switched and Monitor coprocessor extension bits are both set (i.e. TS = 1 and MP = 1).

The T bit in the 386 DX TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1 then upon entry to a new task a debug exception 1 will be generated.

4.4.7 Initialization and Transition to Protected Mode

Since the 386 DX begins executing in Real Mode immediately after RESET it is necessary to initialize the system tables and registers with the appropriate values.

The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256 bytes long, and GDT must contain descriptors for the initial code, and data segments. Figure 4-17 shows the tables and Figure 4-18 the descriptors needed for a simple Protected Mode 386 DX system. It has a single code and single data/stack segment each four gigabytes long and a single privilege level PL = 0.

The actual method of enabling Protected Mode is to load CR0 with the PE bit set, via the MOV CR0, R/M instruction. This puts the 386 DX in Protected Mode.

After enabling Protected Mode, the next instruction should execute an intersegment JMP to load the CS register and flush the instruction decode queue. The final step is to load all of the data segment registers with the initial selector values.

An alternate approach to entering Protected Mode which is especially appropriate for multi-tasking operating systems, is to use the built in task-switch to load all of the registers. In this case the GDT would contain two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode would jump to the TSS causing a task switch and loading all of the registers with the values stored in the TSS. The Task State Segment Register should be initialized to point to a valid TSS descriptor since a task switch saves the state of the current task in a task state segment.

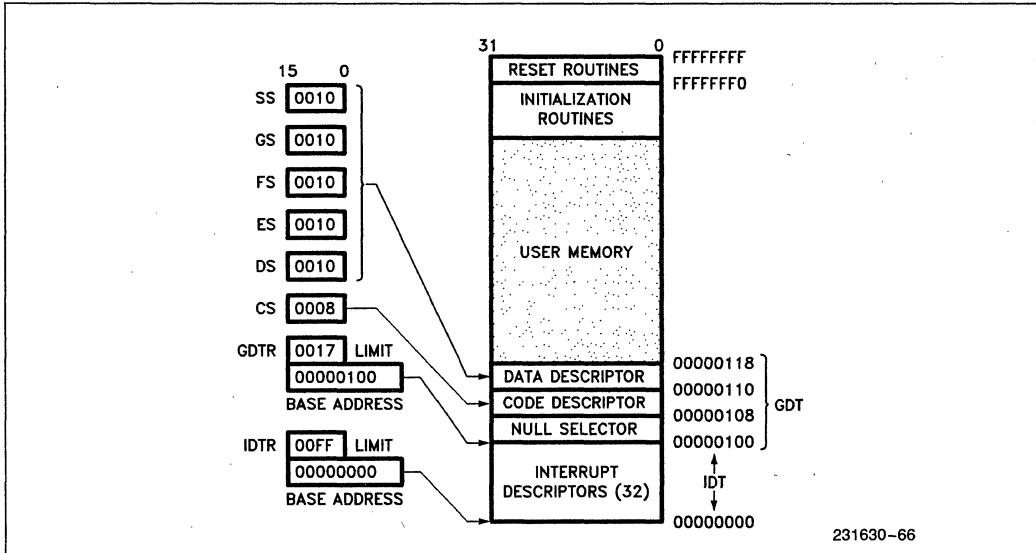


Figure 4-17. Simple Protected System

DATA DESCRIPTOR	SEGMENT BASE 15...0 0118 (H)				SEGMENT LIMIT 15...0 FFFF (H)					
	BASE 31...24 00 (H)	G 1	D 1	00	LIMIT 19.16 F (H)	1	001	0010	BASE 23...16 00 (H)	
CODE DESCRIPTOR	SEGMENT BASE 15...0 0118 (H)				SEGMENT LIMIT 15...0 FFFF (H)					
	BASE 31...24 00 (H)	G 1	D 1	00	LIMIT 19.16 F (H)	1	001	1101	BASE 23...16 00 (H)	
NULL					DESCRIPTOR					
31					24		16 15		8 0	

Figure 4-18. GDT Descriptors for Simple System

4.4.8 Tools for Building Protected Systems

In order to simplify the design of a protected multitasking system, Intel provides a tool which allows the system designer an easy method of constructing the data structures needed for a Protected Mode 386 DX system. This tool is the builder BLD-386™. BLD-386 lets the operating system writer specify all of the segment descriptors discussed in the previous sections (LDTs, IDTs, GDTs, Gates, and TSSs) in a high-level language.

4.5 PAGING

4.5.1 Paging Concepts

Paging is another type of memory management useful for virtual memory multitasking operating systems. Unlike segmentation which modularizes programs and data into variable length segments, paging divides programs into multiple uniform size pages. Pages bear no direct relation to the logical

structure of a program. While segment selectors can be considered the logical “name” of a program module or data structure, a page most likely corresponds to only a portion of a module or data structure.

By taking advantage of the locality of reference displayed by most programs, only a small number of pages from each active task need be in memory at any one moment.

4.5.2 Paging Organization

4.5.2.1 PAGE MECHANISM

The 386 DX uses two levels of tables to translate the linear address (from the segmentation unit) into a physical address. There are three components to the paging mechanism of the 386 DX: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the 386 DX paging mechanism are the same size, namely, 4K bytes. A uniform size for all of the elements simplifies memory allocation and reallocation schemes, since there is no problem with memory fragmentation. Figure 4-19 shows how the paging mechanism works.

4.5.2.2 PAGE DESCRIPTOR BASE REGISTER

CR2 is the Page Fault Linear Address register. It holds the 32-bit linear address which caused the last page fault detected.

CR3 is the Page Directory Physical Base Address Register. It contains the physical starting address of the Page Directory. The lower 12 bits of CR3 are always zero to ensure that the Page Directory is always page aligned. Loading it via a MOV CR3, reg instruction causes the Page Table Entry cache to be flushed, as will a task switch through a TSS which **changes** the value of CR0. (See 4.5.4 Translation Lookaside Buffer).

4.5.2.3 PAGE DIRECTORY

The Page Directory is 4K bytes long and allows up to 1024 Page Directory Entries. Each Page Directory Entry contains the address of the next level of tables, the Page Tables and information about the page table. The contents of a Page Directory Entry are shown in Figure 4-20. The upper 10 bits of the linear address (A22–A31) are used as an index to select the correct Page Directory Entry.

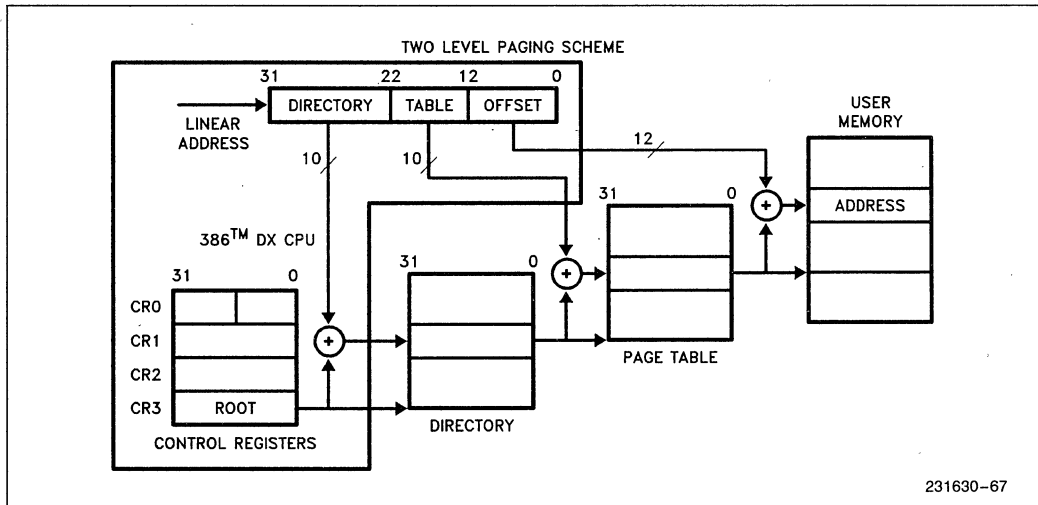


Figure 4-19. Paging Mechanism

	31		12	11	10	9	8	7	6	5	4	3	2	1	0
PAGE TABLE ADDRESS 31..12	OS RESERVED				0	0	D	A	0	0	U	R	—	—	P
											S	W			

Figure 4-20. Page Directory Entry (Points to Page Table)

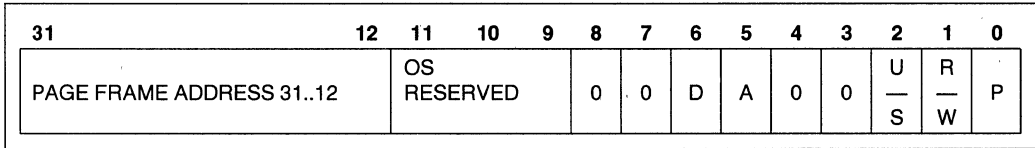


Figure 4-21. Page Table Entry (Points to Page)

4.5.2.4 PAGE TABLES

Each Page Table is 4K bytes and holds up to 1024 Page Table Entries. Page Table Entries contain the starting address of the page frame and statistical information about the page (see Figure 4-21). Address bits A12–A21 are used as an index to select one of the 1024 Page Table Entries. The 20 upper-bit page frame address is concatenated with the lower 12 bits of the linear address to form the physical address. Page tables can be shared between tasks and swapped to disks.

4.5.2.5 PAGE DIRECTORY/TABLE ENTRIES

The lower 12 bits of the Page Table Entries and Page Directory Entries contain statistical information about pages and page tables respectively. The **P** (Present) bit 0 indicates if a Page Directory or Page Table entry can be used in address translation. If P = 1 the entry can be used for address translation; if P = 0 the entry can not be used for translation. Note that the present bit of the page table entry that points to the page where code is currently being executed should always be set. Code that marks its own page not present should not be written. All of the other bits are available for use by the software. For example the remaining 31 bits could be used to indicate where on the disk the page is stored.

The **A** (Accessed) bit 5, is set by the 386 DX for both types of entries before a read or write access occurs to an address covered by the entry. The **D** (Dirty) bit 6 is set to 1 before a write to an address covered by that page table entry occurs. The D bit is undefined for Page Directory Entries. When the P, A and D bits are updated by the 386 DX, the processor generates a Read-Modify-Write cycle which locks the bus and prevents conflicts with other processors or peripherals. Software which modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multi-master systems.

The 3 bits marked **OS Reserved** in Figure 4-20 and Figure 4-21 (bits 9–11) are software definable. OSs are free to use these bits for whatever purpose they wish. An example use of the **OS Reserved** bits would be to store information about page aging. By keeping track of how long a page has been in memory since being accessed, an operating system can implement a page replacement algorithm like Least Recently Used.

The (User/Supervisor) U/S bit 2 and the (Read/Write) R/W bit 1 are used to provide protection attributes for individual pages.

4.5.3 Page Level Protection (R/W, U/S Bits)

The 386 DX provides a set of protection attributes for paging systems. The paging mechanism distinguishes between two levels of protection: User which corresponds to level 3 of the segmentation based protection, and supervisor which encompasses all of the other protection levels (0, 1, 2). Programs executing at Level 0, 1 or 2 bypass the page protection, although segmentation based protection is still enforced by the hardware.

The U/S and R/W bits are used to provide User/Supervisor and Read/Write protection for individual pages or for all pages covered by a Page Table Directory Entry. The U/S and R/W bits in the first level Page Directory Table apply to all pages described by the page table pointed to by that directory entry. The U/S and R/W bits in the second level Page Table Entry apply only to the page described by that entry. The U/S and R/W bits for a given page are obtained by taking the most restrictive of the U/S and R/W from the Page Directory Table Entries and the Page Table Entries and using these bits to address the page.

Example: If the U/S and R/W bits for the Page Directory entry were 10 and the U/S and R/W bits for the Page Table Entry were 01, the access rights for the page would be 01, the numerically smaller of the two. Table 4-4 shows the affect of the U/S and R/W bits on accessing memory.

Table 4-4. Protection Provided by R/W and U/S

U/S	R/W	Permitted Level 3	Permitted Access Levels 0, 1, or 2
0	0	None	Read/Write
0	1	None	Read/Write
1	0	Read-Only	Read/Write
1	1	Read/Write	Read/Write

However a given segment can be easily made read-only for level 0, 1, or 2 via the use of segmented protection mechanisms. (Section 4.4 Protection).

4.5.4 Translation Lookaside Buffer

The 386 DX paging hardware is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the processor was required to access two levels of tables for every memory reference. To solve this problem, the 386 DX keeps a cache of the most recently accessed pages, this cache is called the Translation Lookaside Buffer (TLB). The TLB is a four-way set associative 32-entry page table cache. It automatically keeps the most commonly used Page Table Entries in the processor. The 32-entry TLB coupled with a 4K page size, results in coverage of 128K bytes of memory addresses. For many common multi-tasking systems, the TLB will have a hit rate of about 98%. This means that the processor will only have to access the two-level page structure on 2% of all memory references. Figure 4-22 illustrates how the TLB complements the 386 DX's paging mechanism.

4.5.5 Paging Operation

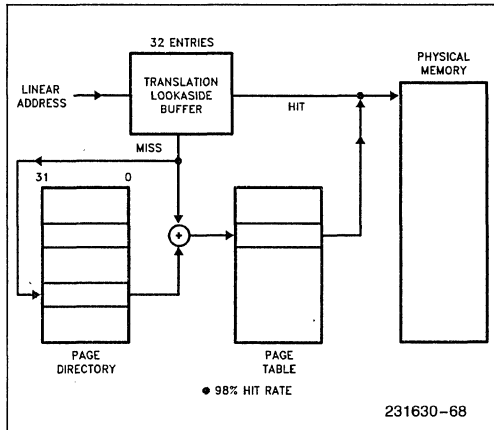


Figure 4-22. Translation Lookaside Buffer

The paging hardware operates in the following fashion. The paging unit hardware receives a 32-bit linear address from the segmentation unit. The upper 20 linear address bits are compared with all 32 entries in the TLB to determine if there is a match. If there is a match (i.e. a TLB hit), then the 32-bit physical address is calculated and will be placed on the address bus.

However, if the page table entry is not in the TLB, the 386 DX will read the appropriate Page Directory Entry. If P = 1 on the Page Directory Entry indicating that the page table is in memory, then the 386 DX will read the appropriate Page Table Entry

and set the Access bit. If P = 1 on the Page Table Entry indicating that the page is in memory, the 386 DX will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. However, if P = 0 for either the Page Directory Entry or the Page Table Entry, then the processor will generate a page fault, an Exception 14.

The processor will also generate an exception 14, page fault, if the memory reference violated the page protection attributes (i.e. U/S or R/W) (e.g. trying to write to a read-only page). CR2 will hold the linear address which caused the page fault. If a second page fault occurs, while the processor is attempting to enter the service routine for the first, then the processor will invoke the page fault (exception 14) handler a second time, rather than the double fault (exception 8) handler. Since Exception 14 is classified as a fault, CS: EIP will point to the instruction causing the page fault. The 16-bit error code pushed as part of the page fault handler will contain status bits which indicate the cause of the page fault.

The 16-bit error code is used by the operating system to determine how to handle the page fault. Figure 4-23A shows the format of the page-fault error code and the interpretation of the bits.

NOTE:

Even though the bits in the error code (U/S, W/R, and P) have similar names as the bits in the Page Directory/Table Entries, the interpretation of the error code bits is different. Figure 4-23B indicates what type of access caused the page fault.

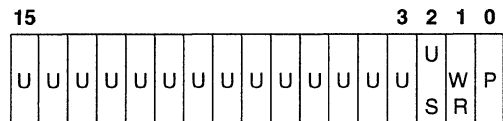


Figure 4-23A. Page Fault Error Code Format

U/S: The U/S bit indicates whether the access causing the fault occurred when the processor was executing in User mode (U/S = 1) or in Supervisor mode (U/S = 0)

W/R: The W/R bit indicates whether the access causing the fault was a Read (W/R = 0) or a Write (W/R = 1)

P: The P bit indicates whether a page fault was caused by a not-present page (P = 0), or by a page level protection violation (P = 1)

U: UNDEFINED

U/S	W/R	Access Type
0	0	Supervisor* Read
0	1	Supervisor Write
1	0	User Read
1	1	User Write

*Descriptor table access will fault with U/S = 0, even if the program is executing at level 3.

Figure 4-23B. Type of Access Causing Page Fault

4.5.6 Operating System Responsibilities

The 386 DX takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for setting up the initial page tables, and handling any page faults. The operating system also is required to invalidate (i.e. flush) the TLB when any changes are made to any of the page table entries. The operating system must reload CR3 to cause the TLB to be flushed.

Setting up the tables is simply a matter of loading CR3 with the address of the Page Directory, and allocating space for the Page Directory and the Page Tables. The primary responsibility of the operating system is to implement a swapping policy and handle all of the page faults.

A final concern of the operating system is to ensure that the TLB cache matches the information in the paging tables. In particular, any time the operating system sets the P present bit of page table entry to zero, the TLB must be flushed. Operating systems may want to take advantage of the fact that CR3 is stored as part of a TSS, to give every task or group of tasks its own set of page tables.

4.6 VIRTUAL 8086 ENVIRONMENT

4.6.1 Executing 8086 Programs

The 386 DX allows the execution of 8086 application programs in both Real Mode and in the Virtual 8086 Mode (Virtual Mode). Of the two methods, Virtual 8086 Mode offers the system designer the most flexibility. The Virtual 8086 Mode allows the execution of 8086 applications, while still allowing the system designer to take full advantage of the 386 DX protection mechanism. In particular, the 386 DX allows the simultaneous execution of 8086 operating systems and its applications, and a 386 DX operating system and both 80286 and 386 DX appli-

cations. Thus, in a multi-user 386 DX computer, one person could be running an MS-DOS spreadsheet, another person using MS-DOS, and a third person could be running multiple Unix utilities and applications. Each person in this scenario would believe that he had the computer completely to himself. Figure 4-24 illustrates this concept.

4.6.2 Virtual 8086 Mode Addressing Mechanism

One of the major differences between 386 DX Real and Protected modes is how the segment selectors are interpreted. When the processor is executing in Virtual 8086 Mode the segment registers are used in an identical fashion to Real Mode. The contents of the segment register is shifted left 4 bits and added to the offset to form the segment base linear address.

The 386 DX allows the operating system to specify which programs use the 8086 style address mechanism, and which programs use Protected Mode addressing, on a per task basis. Through the use of paging, the one megabyte address space of the Virtual Mode task can be mapped to anywhere in the 4 gigabyte linear address space of the 386 DX. Like Real Mode, Virtual Mode effective addresses (i.e., segment offsets) that exceed 64K byte will cause an exception 13. However, these restrictions should not prove to be important, because most tasks running in Virtual 8086 Mode will simply be existing 8086 application programs.

4.6.3 Paging In Virtual Mode

The paging hardware allows the concurrent running of multiple Virtual Mode tasks, and provides protection and operating system isolation. Although it is not strictly necessary to have the paging hardware enabled to run Virtual Mode tasks, it is needed in order to run multiple Virtual Mode tasks or to relocate the address space of a Virtual Mode task to physical address space greater than one megabyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into up to 256 pages. Each one of the pages can be located anywhere within the maximum 4 gigabyte physical address space of the 386 DX. In addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations. Finally, the paging hardware allows the sharing of the 8086 operating

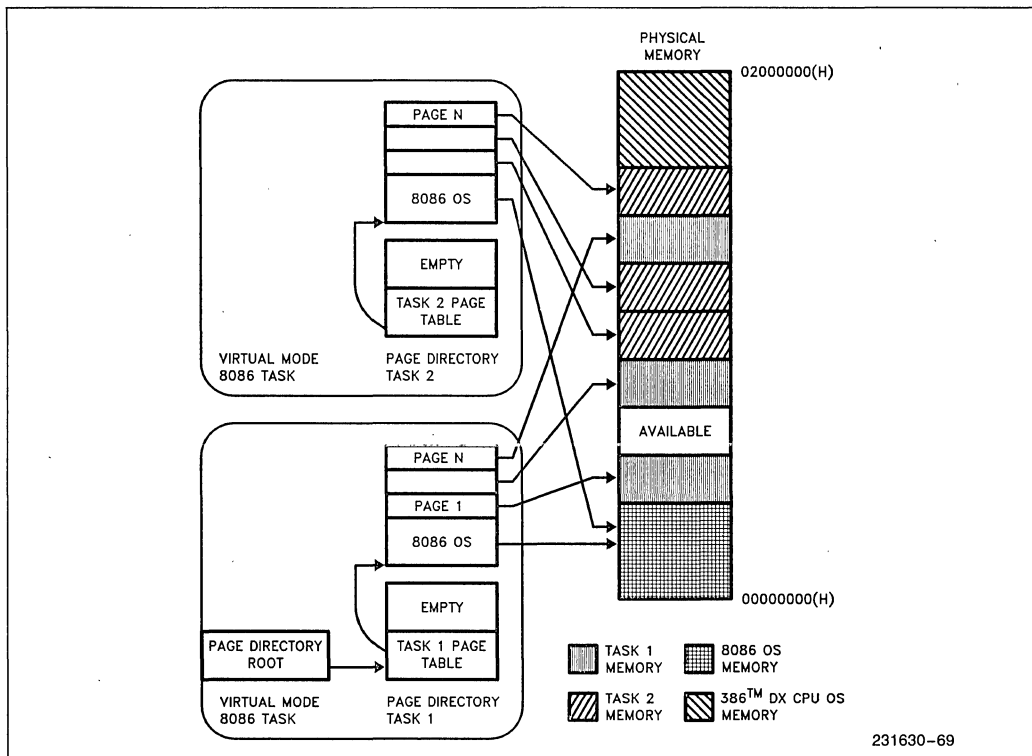


Figure 4-24. Virtual 8086 Environment Memory Management

system code between multiple 8086 applications. Figure 4-24 shows how the 386 DX paging hardware enables multiple 8086 programs to run under a virtual memory demand paged system.

4.6.4 Protection and I/O Permission Bitmap

All Virtual 8086 Mode programs execute at privilege level 3, the level of least privilege. As such, Virtual 8086 Mode programs are subject to all of the protection checks defined in Protected Mode. (This is different from Real Mode which implicitly is executing at privilege level 0, the level of greatest privilege.) Thus, an attempt to execute a privileged instruction when in Virtual 8086 Mode will cause an exception 13 fault.

The following are privileged instructions, which may be executed only at Privilege Level 0. Therefore, attempting to execute these instructions in Virtual 8086 Mode (or anytime CPL > 0) causes an exception 13 fault:

```
LIDT;  MOV DRn,reg;  MOV reg,DRn;
LGDT;  MOV TRn,reg;  MOV reg,TRn;
```

```
LMSW;  MOV CRn,reg;  MOV reg,CRn.
CLTS;
HLT;
```

Several instructions, particularly those applying to the multitasking model and protection model, are available only in Protected Mode. Therefore, attempting to execute the following instructions in Real Mode or in Virtual 8086 Mode generates an exception 6 fault:

```
LTR;   STR;
LLDT;  SLDT;
LAR;   VERR;
LSL;   VERW;
ARPL.
```

The instructions which are IOPL-sensitive in Protected Mode are:

```
IN;    STI;
OUT;   CLI
INS;
OUTS;
REP INS;
REP OUTS;
```

In Virtual 8086 Mode, a slightly different set of instructions are made IOPL-sensitive. The following instructions are IOPL-sensitive in Virtual 8086 Mode:

```
INT n;   STI;
PUSHF;  CLI;
POPF;   IRET
```

The PUSHF, POPF, and IRET instructions are IOPL-sensitive in Virtual 8086 Mode only. This provision allows the IF flag (interrupt enable flag) to be virtualized to the Virtual 8086 Mode program. The INT n software interrupt instruction is also IOPL-sensitive in Virtual 8086 Mode. Note, however, that the INT 3 (opcode 0CCH), INTO, and BOUND instructions are not IOPL-sensitive in Virtual 8086 mode (they aren't IOPL sensitive in Protected Mode either).

Note that the I/O instructions (IN, OUT, INS, OUTS, REP INS, and REP OUTS) are **not** IOPL-sensitive in Virtual 8086 mode. Rather, the I/O instructions become automatically sensitive to the **I/O Permission Bitmap** contained in the **386 DX Task State Segment**. The I/O Permission Bitmap, automatically used by the 386 DX in Virtual 8086 Mode, is illustrated by Figures 4.15a and 4-15b.

The I/O Permission Bitmap can be viewed as a 0–64 Kbit bit string, which begins in memory at offset Bit_Map_Offset in the current TSS. Bit_Map_Offset must be ≤ DFFFH so the entire bit map and the byte FFH which follows the bit map are all at offsets ≤ FFFFH from the TSS base. The 16-bit pointer Bit_Map_Offset (15:0) is found in the word beginning at offset 66H (102 decimal) from the TSS base, as shown in Figure 4-15a.

Each bit in the I/O Permission Bitmap corresponds to a single byte-wide I/O port, as illustrated in Figure 4-15a. If a bit is 0, I/O to the corresponding byte-wide port can occur without generating an exception. Otherwise the I/O instruction causes an exception 13 fault. Since every byte-wide I/O port must be protectable, all bits corresponding to a word-wide or dword-wide port must be 0 for the word-wide or dword-wide I/O to be permitted. If all the referenced bits are 0, the I/O will be allowed. If any referenced bits are 1, the attempted I/O will cause an exception 13 fault.

Due to the use of a pointer to the base of the I/O Permission Bitmap, the bitmap may be located anywhere within the TSS, or may be ignored completely by pointing the Bit_Map_Offset (15:0) beyond the limit of the TSS segment. In the same manner, only a small portion of the 64K I/O space need have an associated map bit, by adjusting the TSS limit to truncate the bitmap. This eliminates the commitment of 8K of memory when a complete bitmap is not required, while allowing the fully general case if desired.

EXAMPLE OF BITMAP FOR I/O PORTS 0–255: Setting the TSS limit to {bit_Map_Offset + 31 + 1**} [** see note below] will allow a 32-byte bitmap for the I/O ports #0–255, plus a terminator byte of all 1's [** see note below]. This allows the I/O bitmap to control I/O Permission to I/O port 0–255 while causing an exception 13 fault on attempted I/O to any I/O port 80256 through 65,565.

****IMPORTANT IMPLEMENTATION NOTE:** Beyond the last byte of I/O mapping information in the I/O Permission Bitmap **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the 386 DX TSS segment (see Figure 4-15a).

4.6.5 Interrupt Handling

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled in a unique fashion. When running in Virtual Mode all interrupts and exceptions involve a privilege change back to the host 386 DX operating system. The 386 DX operating system determines if the interrupt comes from a Protected Mode application or from a Virtual Mode program by examining the VM bit in the EFLAGS image stored on the stack.

When a Virtual Mode program is interrupted and execution passes to the interrupt routine at level 0, the VM bit is cleared. However, the VM bit is still set in the EFLAG image on the stack.

The 386 DX operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The 386 DX operating system may choose to let the 8086 operating system handle the interrupt or it may emulate the function of the interrupt handler. For example, many 8086 operating system calls are accessed by PUSHing parameters on the stack, and then executing an INT n instruction. If the IOPL is set to 0 then all INT n instructions will be intercepted by the 386 DX Microprocessor operating system. The 386 DX operating system could emulate the 8086 operating system's call. Figure 4-25 shows how the 386 DX operating system could intercept an 8086 operating system's call to "Open a File".

A 386 DX operating system can provide a Virtual 8086 Environment which is totally transparent to the application software via intercepting and then emulating 8086 operating system's calls, and intercepting IN and OUT instructions.

4.6.6 Entering and Leaving Virtual 8086 Mode

Virtual 8086 mode is entered by executing an IRET instruction (at CPL = 0), or Task Switch (at any CPL) to a 386 DX task whose 386 DX TSS has a FLAGS image containing a 1 in the VM bit position while the processor is executing in Protected Mode. That is, one way to enter Virtual 8086 mode is to switch to a task with a 386 DX TSS that has a 1 in the VM bit in the EFLAGS image. The other way is to execute a 32-bit IRET instruction at privilege level 0, where the stack has a 1 in the VM bit in the EFLAGS image. POPF does not affect the VM bit, even if the processor is in Protected Mode or level 0, and so cannot be used to enter Virtual 8086 Mode. PUSHF always pushes a 0 in the VM bit, even if the processor is in Virtual 8086 Mode, so that a program cannot tell if it is executing in REAL mode, or in Virtual 8086 mode.

The VM bit can be set by executing an IRET instruction only at privilege level 0, or by any instruction or Interrupt which causes a task switch in Protected Mode (with VM = 1 in the new FLAGS image), and can be cleared only by an interrupt or exception in Virtual 8086 Mode. IRET and POPF instructions executed in REAL mode or Virtual 8086 mode will not change the value in the VM bit.

The transition out of virtual 8086 mode to 386 DX protected mode occurs only on receipt of an interrupt or exception (such as due to a sensitive instruction). In Virtual 8086 mode, all interrupts and exceptions vector through the protected mode IDT, and enter an interrupt handler in protected 386 DX mode. That is, as part of interrupt processing, the VM bit is cleared.

Because the matching IRET must occur from level 0, if an Interrupt or Trap Gate is used to field an interrupt or exception out of Virtual 8086 mode, the Gate must perform an inter-level interrupt only to level 0. Interrupt or Trap Gates through conforming segments, or through segments with DPL > 0, will raise a GP fault with the CS selector as the error code.

4.6.6.1 TASK SWITCHES TO/FROM VIRTUAL 8086 MODE

Tasks which can execute in virtual 8086 mode must be described by a TSS with the new 386 DX format (TYPE 9 or 11 descriptor).

A task switch out of virtual 8086 mode will operate exactly the same as any other task switch out of a task with a 386 DX TSS. All of the programmer visible state, including the FLAGS register with the VM bit set to 1, is stored in the TSS. The segment

registers in the TSS will contain 8086 segment base values rather than selectors.

A task switch into a task described by a 386 DX TSS will have an additional check to determine if the incoming task should be resumed in virtual 8086 mode. Tasks described by 80286 format TSSs cannot be resumed in virtual 8086 mode, so no check is required there (the FLAGS image in 80286 format TSS has only the low order 16 FLAGS bits). Before loading the segment register images from a 386 DX TSS, the FLAGS image is loaded, so that the segment registers are loaded from the TSS image as 8086 segment base values. The task is now ready to resume in virtual 8086 execution mode.

4.6.6.2 TRANSITIONS THROUGH TRAP AND INTERRUPT GATES, AND IRET

A task switch is one way to enter or exit virtual 8086 mode. The other method is to exit through a Trap or Interrupt gate, as part of handling an interrupt, and to enter as part of executing an IRET instruction. The transition out must use a 386 DX Trap Gate (Type 14), or 386 DX Interrupt Gate (Type 15), which must point to a non-conforming level 0 segment (DPL = 0) in order to permit the trap handler to IRET back to the Virtual 8086 program. The Gate must point to a non-conforming level 0 segment to perform a level switch to level 0 so that the matching IRET can change the VM bit. 386 DX gates must be used, since 80286 gates save only the low 16 bits of the FLAGS register, so that the VM bit will not be saved on transitions through the 80286 gates. Also, the 16-bit IRET (presumably) used to terminate the 80286 interrupt handler will pop only the lower 16 bits from FLAGS, and will not affect the VM bit. The action taken for a 386 DX Trap or Interrupt gate if an interrupt occurs while the task is executing in virtual 8086 mode is given by the following sequence.

- (1) Save the FLAGS register in a temp to push later. Turn off the VM and TF bits, and if the interrupt is serviced by an Interrupt Gate, turn off IF also.
- (2) Interrupt and Trap gates must perform a level switch from 3 (where the VM86 program executes) to level 0 (so IRET can return). This process involves a stack switch to the stack given in the TSS for privilege level 0. Save the Virtual 8086 Mode SS and ESP registers to push in a later step. The segment register load of SS will be done as a Protected Mode segment load, since the VM bit was turned off above.
- (3) Push the 8086 segment register values onto the new stack, in the order: GS, FS, DS, ES. These are pushed as 32-bit quantities, with undefined values in the upper 16 bits. Then load these 4 registers with null selectors (0).

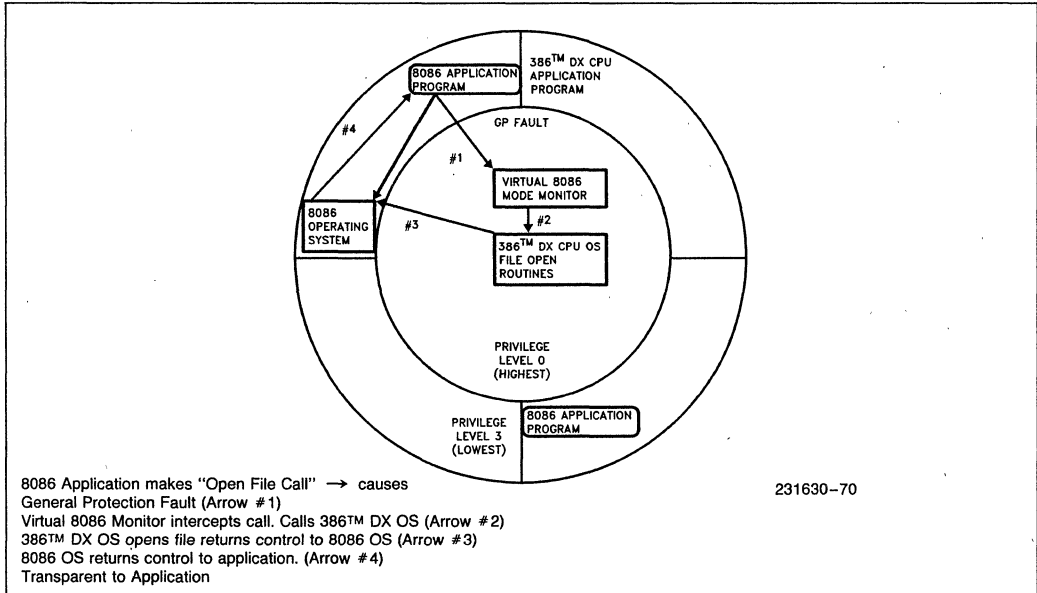


Figure 4-25. Virtual 8086 Environment Interrupt and Call Handling

- (4) Push the old 8086 stack pointer onto the new stack by pushing the SS register (as 32-bits, high bits undefined), then pushing the 32-bit ESP register saved above.
- (5) Push the 32-bit FLAGS register saved in step 1.
- (6) Push the old 8086 instruction pointer onto the new stack by pushing the CS register (as 32-bits, high bits undefined), then pushing the 32-bit EIP register.
- (7) Load up the new CS:EIP value from the interrupt gate, and begin execution of the interrupt routine in protected 386 DX mode.

The transition out of virtual 8086 mode performs a level change and stack switch, in addition to changing back to protected mode. In addition, all of the 8086 segment register images are stored on the stack (behind the SS:ESP image), and then loaded with null (0) selectors before entering the interrupt handler. This will permit the handler to safely save and restore the DS, ES, FS, and GS registers as 80286 selectors. This is needed so that interrupt handlers which don't care about the mode of the interrupted program can use the same prolog and epilog code for state saving (i.e. push all registers in prolog, pop all in epilog) regardless of whether or not a "native" mode or Virtual 8086 mode program was interrupted. Restoring null selectors to these registers before executing the IRET will not cause a trap in the interrupt handler. Interrupt routines which expect values in the segment registers, or return values in segment registers will have to obtain/return values from the 8086 register images pushed onto

the new stack. They will need to know the mode of the interrupted program in order to know where to find/return segment registers, and also to know how to interpret segment register values.

The IRET instruction will perform the inverse of the above sequence. Only the extended 386 DXs IRET instruction (operand size=32) can be used, and must be executed at level 0 to change the VM bit to 1.

- (1) If the NT bit in the FLAGS register is on, an interrupt return is performed. The current state is stored in the current TSS, and the link field in the current TSS is used to locate the TSS for the interrupted task which is to be resumed.

Otherwise, continue with the following sequence.

- (2) Read the FLAGS image from SS:8[ESP] into the FLAGS register. This will set VM to the value active in the interrupted routine.
- (3) Pop off the instruction pointer CS:EIP. EIP is popped first, then a 32-bit word is popped which contains the CS value in the lower 16 bits. If VM=0, this CS load is done as a protected mode segment load. If VM=1, this will be done as an 8086 segment load.
- (4) Increment the ESP register by 4 to bypass the FLAGS image which was "popped" in step 1.
- (5) If VM=1, load segment registers ES, DS, FS, and GS from memory locations SS:[ESP+8], SS:[ESP+12], SS:[ESP+16], and SS:[ESP+20], respectively, where the new val-

ue of ESP stored in step 4 is used. Since VM = 1, these are done as 8086 segment register loads.

Else if VM = 0, check that the selectors in ES, DS, FS, and GS are valid in the interrupted routine. Null out invalid selectors to trap if an attempt is made to access through them.

- (6) If (RPL(CS) > CPL), pop the stack pointer SS:ESP from the stack. The ESP register is popped first, followed by 32-bits containing SS in the lower 16 bits. If VM = 0, SS is loaded as a protected mode segment register load. If VM = 1, an 8086 segment register load is used.
- (7) Resume execution of the interrupted routine. The VM bit in the FLAGS register (restored from the interrupt routine's stack image in step 1) determines whether the processor resumes the interrupted routine in Protected mode of Virtual 8086 mode.

5. FUNCTIONAL DATA

5.1 INTRODUCTION

The 386 DX features a straightforward functional interface to the external hardware. The 386 DX has separate, parallel buses for data and address. The data bus is 32-bits in width, and bidirectional. The address bus outputs 32-bit address values in the most directly usable form for the high-speed local bus: 4 individual byte enable signals, and the 30 upper-order bits as a binary value. The data and address buses are interpreted and controlled with their associated control signals.

A **dynamic data bus sizing** feature allows the processor to handle a mix of 32- and 16-bit external buses on a cycle-by-cycle basis (see **5.3.4 Data Bus Sizing**). If 16-bit bus size is selected, the 386 DX automatically makes any adjustment needed, even performing another 16-bit bus cycle to complete the transfer if that is necessary. 8-bit peripheral devices may be connected to 32-bit or 16-bit buses with no loss of performance. A **new address pipelining option** is provided and applies to 32-bit and 16-bit buses for substantially improved memory utilization, especially for the most heavily used memory resources.

The **address pipelining option**, when selected, typically allows a given memory interface to operate with one less wait state than would otherwise be required (see **5.4.2 Address Pipelining**). The pipelined bus is also well suited to interleaved memory designs. When address pipelining is requested by the external hardware, the 386 DX will output the address and bus cycle definition of the next bus cycle (if it is internally available) even while waiting for the current cycle to be acknowledged.

Non-pipelined address timing, however, is ideal for external cache designs, since the cache memory will typically be fast enough to allow non-pipelined cycles. For maximum design flexibility, the address pipelining option is selectable on a cycle-by-cycle basis.

The processor's bus cycle is the basic mechanism for information transfer, either from system to processor, or from processor to system. 386 DX bus cycles perform data transfer in a minimum of only two clock periods. On a 32-bit data bus, the maximum 386 DX transfer bandwidth at 20 MHz is therefore 40 MBytes/sec, at 25 MHz bandwidth, is 50 Mbytes/sec, and at 33 MHz bandwidth, is 66 Mbytes/sec. Any bus cycle will be extended for more than two clock periods, however, if external hardware withholds acknowledgement of the cycle. At the appropriate time, acknowledgement is signalled by asserting the 386 DX READY# input.

The 386 DX can relinquish control of its local buses to allow mastership by other devices, such as direct memory access channels. When relinquished, HLDA is the only output pin driven by the 386 DX providing near-complete isolation of the processor from its system. The near-complete isolation characteristic is ideal when driving the system from test equipment, and in fault-tolerant applications.

Functional data covered in this chapter describes the processor's hardware interface. First, the set of signals available at the processor pins is described (see **5.2 Signal Description**). Following that are the signal waveforms occurring during bus cycles (see **5.3 Bus Transfer Mechanism**, **5.4 Bus Functional Description** and **5.5 Other Functional Descriptions**).

5.2 SIGNAL DESCRIPTION

5.2.1 Introduction

Ahead is a brief description of the 386 DX input and output signals arranged by functional groups. Note the # symbol at the end of a signal name indicates the active, or asserted, state occurs when the signal is at a low voltage. When no # is present after the signal name, the signal is asserted when at the high voltage level.

Example signal: M/IO# — High voltage indicates
Memory selected
— Low voltage indicates
I/O selected

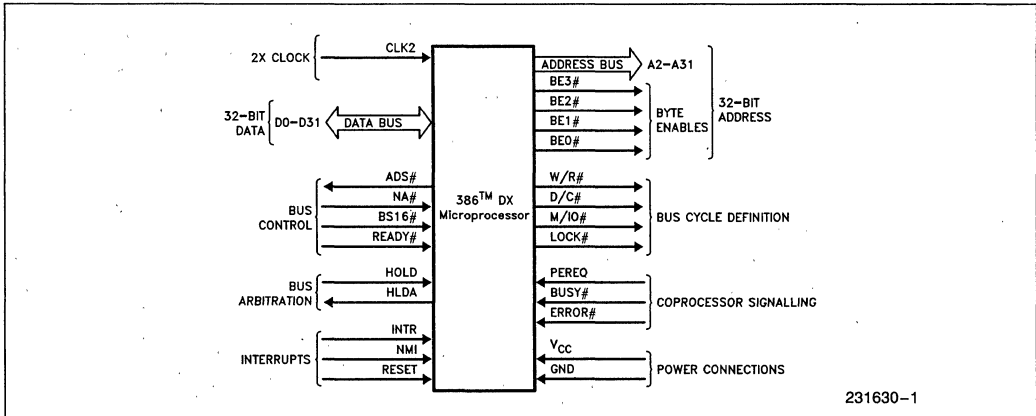


Figure 5-1. Functional Signal Groups

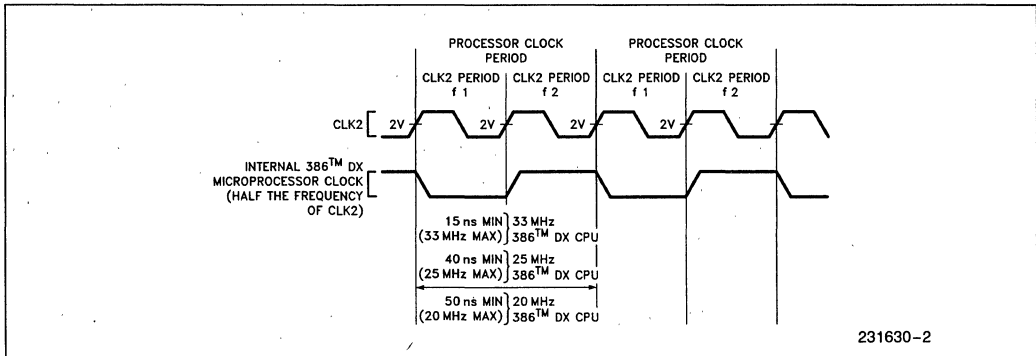


Figure 5-2. CLK2 Signal and Internal Processor Clock

The signal descriptions sometimes refer to AC timing parameters, such as “ t_{25} Reset Setup Time” and “ t_{26} Reset Hold Time.” The values of these parameters can be found in Tables 7-4 and 7-5.

5.2.2 Clock (CLK2)

CLK2 provides the fundamental timing for the 386 DX. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock is comprised of two phases, “phase one” and “phase two.” Each CLK2 period is a phase of the internal clock. Figure 5-2 illustrates the relationship. If desired, the phase of the internal processor clock can be synchronized to a known phase by ensuring the RESET signal falling edge meets its applicable setup and hold times, t_{25} and t_{26} .

5.2.3 Data Bus (D0 through D31)

These three-state bidirectional signals provide the general purpose data path between the 386 DX and

other devices. Data bus inputs and outputs indicate “1” when HIGH. The data bus can transfer data on 32- and 16-bit buses using a data bus sizing feature controlled by the BS16# input. See section 5.2.6 Bus Control. Data bus reads require that read data setup and hold times t_{21} and t_{22} be met for correct operation. In addition, the 386 DX requires that all data bus pins be at a valid logic state (high or low) at the end of each read cycle, when READY# is asserted. During any write operation (and during halt cycles and shutdown cycles), the 386 DX always drives all 32 signals of the data bus even if the current bus size is 16-bits.

5.2.4 Address Bus (BE0# through BE3#, A2 through A31)

These three-state outputs provide physical memory addresses or I/O port addresses. The address bus is capable of addressing 4 gigabytes of physical memory space (00000000H through FFFFFFFFH), and 64 kilobytes of I/O address space (00000000H through 0000FFFFH) for programmed I/O. I/O

transfers automatically generated for 386 DX-to-co-processor communication use I/O addresses 800000F8H through 800000FFH, so A31 HIGH in conjunction with M/IO# LOW allows simple generation of the coprocessor select signal.

The Byte Enable outputs, BE0#–BE3#, directly indicate which bytes of the 32-bit data bus are involved with the current transfer. This is most convenient for external hardware.

- BE0# applies to D0–D7
- BE1# applies to D8–D15
- BE2# applies to D16–D23
- BE3# applies to D24–D31

The number of Byte Enables asserted indicates the physical size of the operand being transferred (1, 2, 3, or 4 bytes). Refer to section 5.3.6 **Operand Alignment**.

When a memory write cycle or I/O write cycle is in progress, and the operand being transferred occupies **only** the upper 16 bits of the data bus (D16–D31), duplicate data is simultaneously presented on the corresponding lower 16-bits of the data bus (D0–D15). This duplication is performed for optimum write performance on 16-bit buses. The pattern of write data duplication is a function of the Byte Enables asserted during the write cycle. Table 5-1 lists the write data present on D0–D31, as a function of the asserted Byte Enable outputs BE0#–BE3#.

5.2.5 Bus Cycle Definition Signals (W/R#, D/C#, M/IO#, LOCK#)

These three-state outputs define the type of bus cycle being performed. W/R# distinguishes between write and read cycles. D/C# distinguishes between data and control cycles. M/IO# distinguishes between memory and I/O cycles. LOCK# distinguishes between locked and unlocked bus cycles.

The primary bus cycle definition signals are W/R#, D/C# and M/IO#, since these are the signals driven valid as the ADS# (Address Status output) is driven asserted. The LOCK# is driven valid at the same time as the first locked bus cycle begins, which due to address pipelining, could be later than ADS# is driven asserted. See 5.4.3.4 **Pipelined Address**. The LOCK# is negated when the READY# input terminates the last bus cycle which was locked.

Exact bus cycle definitions, as a function of W/R#, D/C#, and M/IO#, are given in Table 5-2. Note one combination of W/R#, D/C# and M/IO# is never given when ADS# is asserted (however, that combination, which is listed as “does not occur,” may occur during **idle** bus states when ADS# is **not** asserted). If M/IO#, D/C#, and W/R# are qualified by ADS# asserted, then a decoding scheme may be simplified by using this definition of the “does not occur” combination.

Table 5-1. Write Data Duplication as a Function of BE0#–BE3#

386™ DX Byte Enables				386™ DX Write Data				Automatic Duplication?
BE3#	BE2#	BE1#	BE0#	D24–D31	D16–D23	D8–D15	D0–D7	
High	High	High	Low	undef	undef	undef	A	No
High	High	Low	High	undef	undef	B	undef	No
High	Low	High	High	undef	C	undef	C	Yes
Low	High	High	High	D	undef	D	undef	Yes
High	High	Low	Low	undef	undef	B	A	No
High	Low	Low	High	undef	C	B	undef	No
Low	Low	High	High	D	C	D	C	Yes
High	Low	Low	Low	undef	C	B	A	No
Low	Low	Low	High	D	C	B	undef	No
Low	Low	Low	Low	D	C	B	A	No

Key:

- D = logical write data d24–d31
- C = logical write data d16–d23
- B = logical write data d8–d15
- A = logical write data d0–d7

Table 5-2. Bus Cycle Definition

M/IO#	D/C#	W/R#	Bus Cycle Type	Locked?	
Low	Low	Low	INTERRUPT ACKNOWLEDGE	Yes	
Low	Low	High	does not occur	—	
Low	High	Low	I/O DATA READ	No	
Low	High	High	I/O DATA WRITE	No	
High	Low	Low	MEMORY CODE READ	No	
High	Low	High	HALT: Address = 2 (BE0# High BE1# High BE2# Low BE3# High A2–A31 Low)	SHUTDOWN: Address = 0 (BE0# Low BE1# High BE2# High BE3# High A2–A31 Low)	No
High	High	Low	MEMORY DATA READ	Some Cycles	
High	High	High	MEMORY DATA WRITE	Some Cycles	

5.2.6 Bus Control Signals (ADS#, READY#, NA#, BS16#)

5.2.6.1 INTRODUCTION

The following signals allow the processor to indicate when a bus cycle has begun, and allow other system hardware to control address pipelining, data bus width and bus cycle termination.

5.2.6.2 ADDRESS STATUS (ADS#)

This three-state output indicates that a valid bus cycle definition, and address (W/R#, D/C#, M/IO#, BE0#–BE3#, and A2–A31) is being driven at the 386 DX pins. It is asserted during T1 and T2P bus states (see 5.4.3.2 **Non-pipelined Address** and 5.4.3.4 **Pipelined Address** for additional information on bus states).

5.2.6.3 TRANSFER ACKNOWLEDGE (READY#)

This input indicates the current bus cycle is complete, and the active bytes indicated by BE0#–BE3# and BS16# are accepted or provided. When READY# is sampled asserted during a read cycle or interrupt acknowledge cycle, the 386 DX latches the input data and terminates the cycle. When READY# is sampled asserted during a write cycle, the processor terminates the bus cycle.

READY# is ignored on the first bus state of all bus cycles, and sampled each bus state thereafter until asserted. READY# must eventually be asserted to acknowledge every bus cycle, including Halt Indication and Shutdown Indication bus cycles. When being sampled, READY# must always meet setup and

hold times t_{19} and t_{20} for correct operation. See all sections of 5.4 **Bus Functional Description**.

5.2.6.4 NEXT ADDRESS REQUEST (NA#)

This is used to request address pipelining. This input indicates the system is prepared to accept new values of BE0#–BE3#, A2–A31, W/R#, D/C# and M/IO# from the 386 DX even if the end of the current cycle is not being acknowledged on READY#. If this input is asserted when sampled, the next address is driven onto the bus; provided the next bus request is already pending internally. See 5.4.2 **Address Pipelining** and 5.4.3 **Read and Write Cycles**. NA# must always meet setup and hold times, t_{15} and t_{16} , for correct operation.

5.2.6.5 BUS SIZE 16 (BS16#)

The BS16# feature allows the 386 DX to directly connect to 32-bit and 16-bit data buses. Asserting this input constrains the current bus cycle to use only the lower-order half (D0–D15) of the data bus, corresponding to BE0# and BE1#. Asserting BS16# has no additional effect if only BE0# and/or BE1# are asserted in the current cycle. However, during bus cycles asserting BE2# or BE3#, asserting BS16# will automatically cause the 386 DX to make adjustments for correct transfer of the upper bytes(s) using only physical data signals D0–D15.

If the operand spans both halves of the data bus and BS16# is asserted, the 386 DX will automatically perform another 16-bit bus cycle. BS16# must always meet setup and hold times t_{17} and t_{18} for correct operation.

386 DX I/O cycles are automatically generated for coprocessor communication. Since the 386 DX must transfer 32-bit quantities between itself and the 387 DX, $BS16\#$ *must not* be asserted during 387 DX communication cycles.

5.2.7 Bus Arbitration Signals (HOLD, HLDA)

5.2.7.1 INTRODUCTION

This section describes the mechanism by which the processor relinquishes control of its local buses when requested by another bus master device. See **5.5.1 Entering and Exiting Hold Acknowledge** for additional information.

5.2.7.2 BUS HOLD REQUEST (HOLD)

This input indicates some device other than the 386 DX requires bus mastership.

HOLD must remain asserted as long as any other device is a local bus master. HOLD is not recognized while RESET is asserted. If RESET is asserted while HOLD is asserted, RESET has priority and places the bus into an idle state, rather than the hold acknowledge (high impedance) state.

HOLD is level-sensitive and is a synchronous input. HOLD signals must always meet setup and hold times t_{23} and t_{24} for correct operation.

5.2.7.3 BUS HOLD ACKNOWLEDGE (HLDA)

Assertion of this output indicates the 386 DX has relinquished control of its local bus in response to HOLD asserted, and is in the bus Hold Acknowledge state.

The Hold Acknowledge state offers near-complete signal isolation. In the Hold Acknowledge state, HLDA is the only signal being driven by the 386 DX. The other output signals or bidirectional signals ($D0-D31$, $BE0\#-BE3\#$, $A2-A31$, $W/R\#$, $D/C\#$, $M/I/O\#$, $LOCK\#$ and $ADS\#$) are in a high-impedance state so the requesting bus master may control them. Pullup resistors may be desired on several signals to avoid spurious activity when no bus master is driving them. See **7.2.3 Resistor Recommendations**. Also, one rising edge occurring on the NMI input during Hold Acknowledge is remembered, for processing after the HOLD input is negated.

In addition to the normal usage of Hold Acknowledge with DMA controllers or master peripherals,

the near-complete isolation has particular attractiveness during system test when test equipment drives the system, and in hardware-fault-tolerant applications.

5.2.8 Coprocessor Interface Signals (PEREQ, BUSY #, ERROR #)

5.2.8.1 INTRODUCTION

In the following sections are descriptions of signals dedicated to the numeric coprocessor interface. In addition to the data bus, address bus, and bus cycle definition signals, these following signals control communication between the 386 DX and its 387 DX processor extension.

5.2.8.2 COPROCESSOR REQUEST (PEREQ)

When asserted, this input signal indicates a coprocessor request for a data operand to be transferred to/from memory by the 386 DX. In response, the 386 DX transfers information between the coprocessor and memory. Because the 386 DX has internally stored the coprocessor opcode being executed, it performs the requested data transfer with the correct direction and memory address.

PEREQ is level-sensitive and is allowed to be asynchronous to the CLK2 signal.

5.2.8.3 COPROCESSOR BUSY (BUSY #)

When asserted, this input indicates the coprocessor is still executing an instruction, and is not yet able to accept another. When the 386 DX encounters any coprocessor instruction which operates on the numeric stack (e.g. load, pop, or arithmetic operation), or the WAIT instruction, this input is first automatically sampled until it is seen to be negated. This sampling of the BUSY # input prevents overrunning the execution of a previous coprocessor instruction.

The FNINIT and FNCLEX coprocessor instructions are allowed to execute even if BUSY # is asserted, since these instructions are used for coprocessor initialization and exception-clearing.

BUSY # is level-sensitive and is allowed to be asynchronous to the CLK2 signal.

BUSY # serves an additional function. If BUSY # is sampled LOW at the falling edge of RESET, the 386 DX performs an internal self-test (see **5.5.3 Bus Activity During and Following Reset**). If BUSY # is sampled HIGH, no self-test is performed.

5.2.8.4 COPROCESSOR ERROR (ERROR#)

This input signal indicates that the previous coprocessor instruction generated a coprocessor error of a type not masked by the coprocessor's control register. This input is automatically sampled by the 386 DX when a coprocessor instruction is encountered, and if asserted, the 386 DX generates exception 16 to access the error-handling software.

Several coprocessor instructions, generally those which clear the numeric error flags in the coprocessor or save coprocessor state, do execute without the 386 DX generating exception 16 even if ERROR# is asserted. These instructions are FNINIT, FNCLEX, FSTSW, FSTSWAX, FSTCW, FSTENV, FSAVE, FESTENV and FESAVE.

ERROR# is level-sensitive and is allowed to be asynchronous to the CLK2 signal.

5.2.9 Interrupt Signals (INTR, NMI, RESET)

5.2.9.1 INTRODUCTION

The following descriptions cover inputs that can interrupt or suspend execution of the processor's current instruction stream.

5.2.9.2 MASKABLE INTERRUPT REQUEST (INTR)

When asserted, this input indicates a request for interrupt service, which can be masked by the 386 DX Flag Register IF bit. When the 386 DX responds to the INTR input, it performs two interrupt acknowledge bus cycles, and at the end of the second, latches an 8-bit interrupt vector on D0–D7 to identify the source of the interrupt.

INTR is level-sensitive and is allowed to be asynchronous to the CLK2 signal. To assure recognition of an INTR request, INTR should remain asserted until the first interrupt acknowledge bus cycle begins.

5.2.9.3 NON-MASKABLE INTERRUPT REQUEST (NMI)

This input indicates a request for interrupt service, which cannot be masked by software. The non-

maskable interrupt request is always processed according to the pointer or gate in slot 2 of the interrupt table. Because of the fixed NMI slot assignment, no interrupt acknowledge cycles are performed when processing NMI.

NMI is rising edge-sensitive and is allowed to be asynchronous to the CLK2 signal. To assure recognition of NMI, it must be negated for at least eight CLK2 periods, and then be asserted for at least eight CLK2 periods.

Once NMI processing has begun, no additional NMI's are processed until after the next IRET instruction, which is typically the end of the NMI service routine. If NMI is re-asserted prior to that time, however, one rising edge on NMI will be remembered for processing after executing the next IRET instruction.

5.2.9.4 RESET (RESET)

This input signal suspends any operation in progress and places the 386 DX in a known reset state. The 386 DX is reset by asserting RESET for 15 or more CLK2 periods (80 or more CLK2 periods before requesting self test). When RESET is asserted, all other input pins are ignored, and all other bus pins are driven to an idle bus state as shown in Table 5-3. If RESET and HOLD are both asserted at a point in time, RESET takes priority even if the 386 DX was in a Hold Acknowledge state prior to RESET asserted.

RESET is level-sensitive and must be synchronous to the CLK2 signal. If desired, the phase of the internal processor clock, and the entire 386 DX state can be completely synchronized to external circuitry by ensuring the RESET signal falling edge meets its applicable setup and hold times, t_{25} and t_{26} .

Table 5-3. Pin State (Bus Idle) During Reset

Pin Name	Signal Level During Reset
ADS#	High
D0–D31	High Impedance
BE0#–BE3#	Low
A2–A31	High
W/R#	Low
D/C#	High
M/IO#	Low
LOCK#	High
HLDA	Low

5.2.10 Signal Summary

Table 5-4 summarizes the characteristics of all 386 DX signals.

Table 5-4. 386™ DX Signal Summary

Signal Name	Signal Function	Active State	Input/Output	Input Synchron or Asynchron to CLK2	Output High Impedance During HLDA?
CLK2	Clock	—	I	—	—
D0–D31	Data Bus	High	I/O	S	Yes
BE0#–BE3#	Byte Enables	Low	O	—	Yes
A2–A31	Address Bus	High	O	—	Yes
W/R#	Write-Read Indication	High	O	—	Yes
D/C#	Data-Control Indication	High	O	—	Yes
M/IO#	Memory-I/O Indication	High	O	—	Yes
LOCK#	Bus Lock Indication	Low	O	—	Yes
ADS#	Address Status	Low	O	—	Yes
NA#	Next Address Request	Low	I	S	—
BS16#	Bus Size 16	Low	I	S	—
READY#	Transfer Acknowledge	Low	I	S	—
HOLD	Bus Hold Request	High	I	S	—
HLDA	Bus Hold Acknowledge	High	O	—	No
PEREQ	Coprocessor Request	High	I	A	—
BUSY#	Coprocessor Busy	Low	I	A	—
ERROR#	Coprocessor Error	Low	I	A	—
INTR	Maskable Interrupt Request	High	I	A	—
NMI	Non-Maskable Intrpt Request	High	I	A	—
RESET	Reset	High	I	S	—

5.3 BUS TRANSFER MECHANISM

5.3.1 Introduction

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte, word and double-word lengths may be transferred without restrictions on physical address alignment. Any byte boundary may be used, although two or even three physical bus cycles are performed as required for unaligned operand transfers. See **5.3.4 Dynamic Data Bus Sizing** and **5.3.6 Operand Alignment**.

The 386 DX address signals are designed to simplify external system hardware. Higher-order address bits are provided by A2–A31. Lower-order address in the form of BE0#–BE3# directly provides linear selects for the four bytes of the 32-bit data bus. Physical operand size information is thereby implicitly provided each bus cycle in the most usable form.

Byte Enable outputs BE0#–BE3# are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 5-5. During a bus cycle, any possible pattern of contiguous, asserted Byte Enable outputs can occur, but never patterns having a negated Byte Enable separating two or three asserted Enables.

Address bits A0 and A1 of the physical operand's base address can be created when necessary (for instance, for MULTIBUS® I or MULTIBUS® II interface), as a function of the lowest-order asserted Byte Enable. This is shown by Table 5-6. Logic to generate A0 and A1 is given by Figure 5-3.

Table 5-5. Byte Enables and Associated Data and Operand Bytes

Byte Enable Signal	Associated Data Bus Signals
BE0#	D0–D7 (byte 0—least significant)
BE1#	D8–D15 (byte 1)
BE2#	D16–D23 (byte 2)
BE3#	D24–D31 (byte 3—most significant)

Table 5-6. Generating A0–A31 from BE0#–BE3# and A2–A31

386™ DX Address Signals							
A31	A2		BE3#	BE2#	BE1#	BE0#
Physical Base Address							
A31	A2	A1	A0			
A31	A2	0	0	X	X	Low
A31	A2	0	1	X	X	Low High
A31	A2	1	0	X	Low	High High
A31	A2	1	1	Low	High	High High

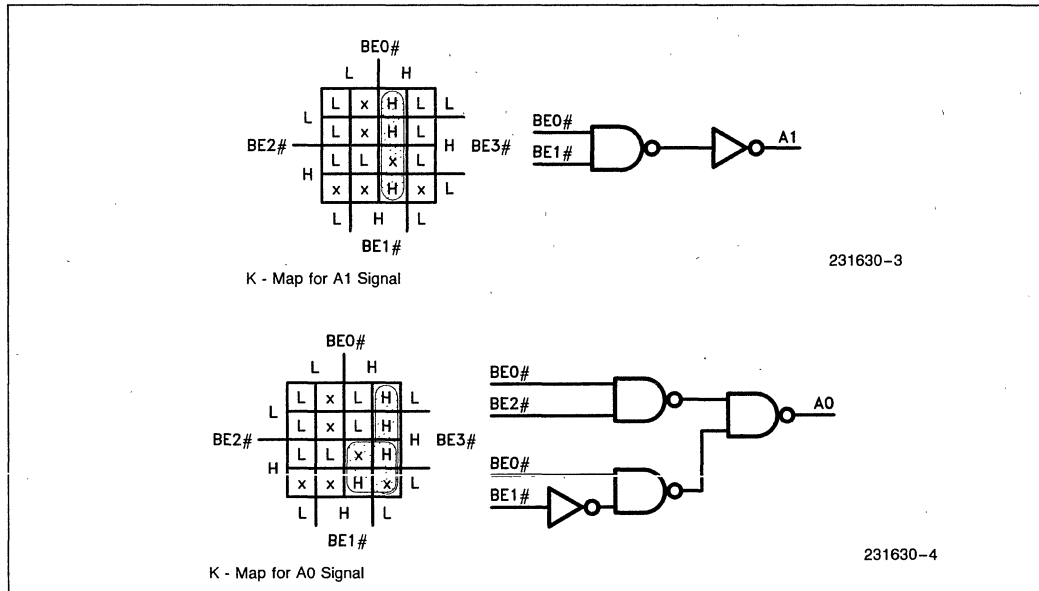


Figure 5-3. Logic to Generate A0, A1 from BE0#–BE3#

Each bus cycle is composed of at least two bus states. Each bus state requires one processor clock period. Additional bus states added to a single bus cycle are called wait states. See **5.4 Bus Functional Description**.

Since a bus cycle requires a minimum of two bus states (equal to two processor clock periods), data can be transferred between external devices and the 386 DX at a maximum rate of one 4-byte Dword every two processor clock periods, for a maximum bus bandwidth of 66 megabytes/second (386 DX operating at 33 MHz processor clock rate).

5.3.2 Memory and I/O Spaces

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. As shown in Figure 5-4, physical memory addresses range from 00000000H to FFFFFFFFH (4 gigabytes) and I/O addresses from 00000000H to 0000FFFFH (64 kilobytes) for programmed I/O. Note the I/O addresses used by the automatic I/O cycles for coprocessor communication are 800000F8H to 800000FFH, beyond the address range of programmed I/O, to allow easy generation of a coprocessor chip select signal using the A31 and M/IO# signals.

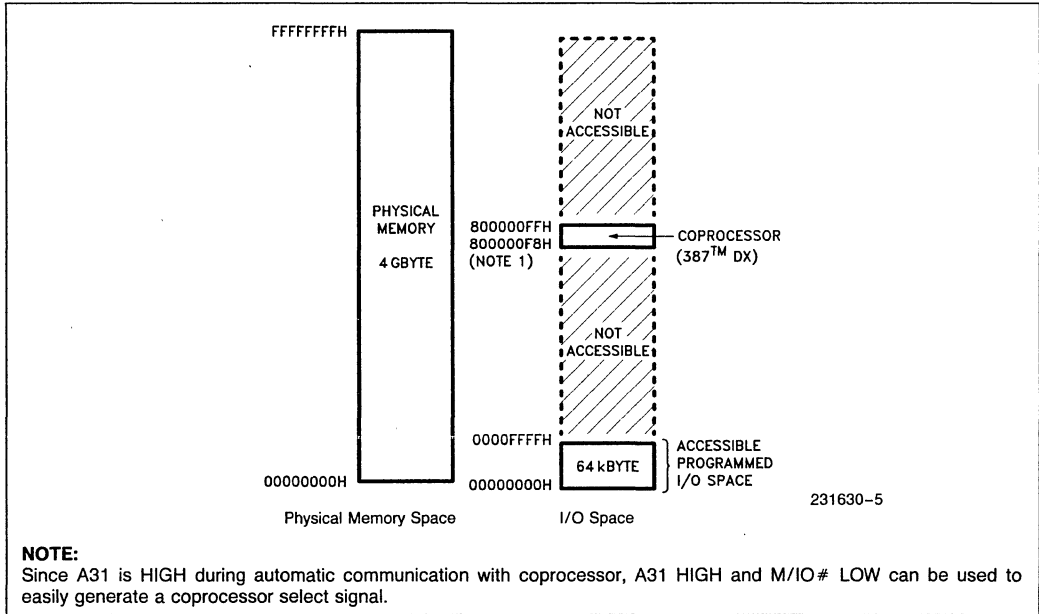


Figure 5-4. Physical Memory and I/O Spaces

5.3.3 Memory and I/O Organization

The 386 DX datapath to memory and I/O spaces can be 32 bits wide or 16 bits wide. When 32-bits wide, memory and I/O spaces are organized naturally as arrays of physical 32-bit Dwords. Each memory or I/O Dword has four individually addressable bytes at consecutive byte addresses. The lowest-addressed byte is associated with data signals D0–D7; the highest-addressed byte with D24–D31.

The 386 DX includes a bus control input, BS16#, that also allows direct connection to 16-bit memory or I/O spaces organized as a sequence of 16-bit words. Cycles to 32-bit and 16-bit memory or I/O devices may occur in any sequence, since the BS16# control is sampled during each bus cycle. See 5.3.4 Dynamic Data Bus Sizing. The Byte Enable signals, BE0#–BE3#, allow byte granularity when addressing any memory or I/O structure, whether 32 or 16 bits wide.

5.3.4 Dynamic Data Bus Sizing

Dynamic data bus sizing is a feature allowing direct processor connection to 32-bit or 16-bit data buses for memory or I/O. A single processor may connect to both size buses. Transfers to or from 32- or 16-bit ports are supported by dynamically determining the bus width during each bus cycle. During each bus cycle an address decoding circuit or the slave de-

vice itself may assert BS16# for 16-bit ports, or negate BS16# for 32-bit ports.

With BS16# asserted, the processor automatically converts operand transfers larger than 16 bits, or misaligned 16-bit transfers, into two or three transfers as required. All operand transfers physically occur on D0–D15 when BS16# is asserted. Therefore, 16-bit memories or I/O devices only connect on data signals D0–D15. No extra transceivers are required.

Asserting BS16# only affects the processor when BE2# and/or BE3# are asserted during the current cycle. If only D0–D15 are involved with the transfer, asserting BS16# has no effect since the transfer can proceed normally over a 16-bit bus whether BS16# is asserted or not. In other words, asserting BS16# has no effect when only the lower half of the bus is involved with the current cycle.

There are two types of situations where the processor is affected by asserting BS16#, depending on which Byte Enables are asserted during the current bus cycle:

Upper Half Only:

Only BE2# and/or BE3# asserted.

Upper and Lower Half:

At least BE1#, BE2# asserted (and perhaps also BE0# and/or BE3#).

Effect of asserting BS16# during "upper half only" read cycles:

Asserting BS16# during "upper half only" reads causes the 386 DX to read data on the lower 16 bits of the data bus and ignore data on the upper 16 bits of the data bus. Data that would have been read from D16–D31 (as indicated by BE2# and BE3#) will instead be read from D0–D15 respectively.

Effect of asserting BS16# during "upper half only" write cycles:

Asserting BS16# during "upper half only" writes does not affect the 386 DX. When only BE2# and/or BE3# are asserted during a write cycle the 386 DX always duplicates data signals D16–D31 onto D0–D15 (see Table 5-1). Therefore, no further 386 DX action is required to perform these writes on 32-bit or 16-bit buses.

Effect of asserting BS16# during "upper and lower half" read cycles:

Asserting BS16# during "upper and lower half" reads causes the processor to perform two 16-bit read cycles for complete physical operand transfer. Bytes 0 and 1 (as indicated by BE0# and BE1#) are read on the first cycle using D0–D15. Bytes 2 and 3 (as indicated by BE2# and BE3#) are read during the second cycle, again using D0–D15. D16–D31 are ignored during both 16-bit cycles. BE0# and BE1# are always negated during the second 16-bit cycle (See Figure 5-14, cycles 2 and 2a).

Effect of asserting BS16# during "upper and lower half" write cycles:

Asserting BS16# during "upper and lower half" writes causes the 386 DX to perform two 16-bit write cycles for complete physical operand transfer. All bytes are available the first write cycle allowing external hardware to receive Bytes 0 and 1 (as indicated by BE0# and BE1#) using D0–D15. On the second cycle the 386 DX duplicates Bytes 2 and 3 on D0–D15 and Bytes 2 and 3 (as indicated by BE2# and BE3#) are written using D0–D15. BE0# and BE1# are always negated during the second 16-bit cycle. BS16# must be asserted during the second 16-bit cycle. See Figure 5-14, cycles 1 and 1a.

5.3.5 Interfacing with 32- and 16-Bit Memories

In 32-bit-wide physical memories such as Figure 5-5, each physical Dword begins at a byte address that is a multiple of 4. A2–A31 are directly used as a Dword select and BE0#–BE3# as byte selects. BS16# is negated for all bus cycles involving the 32-bit array.

When 16-bit-wide physical arrays are included in the system, as in Figure 5-6, each 16-bit physical word begins at a address that is a multiple of 2. Note the address is decoded, to assert BS16# only during bus cycles involving the 16-bit array. (If desiring to

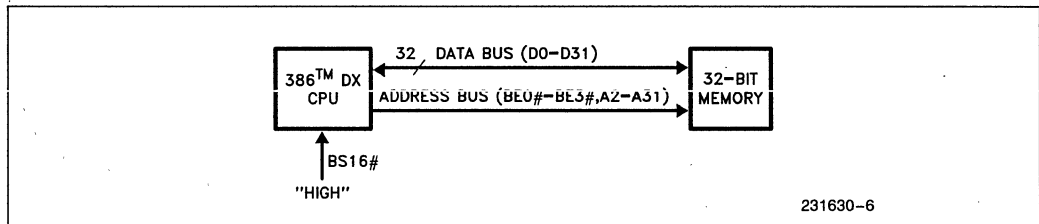


Figure 5-5. 386™ DX with 32-Bit Memory

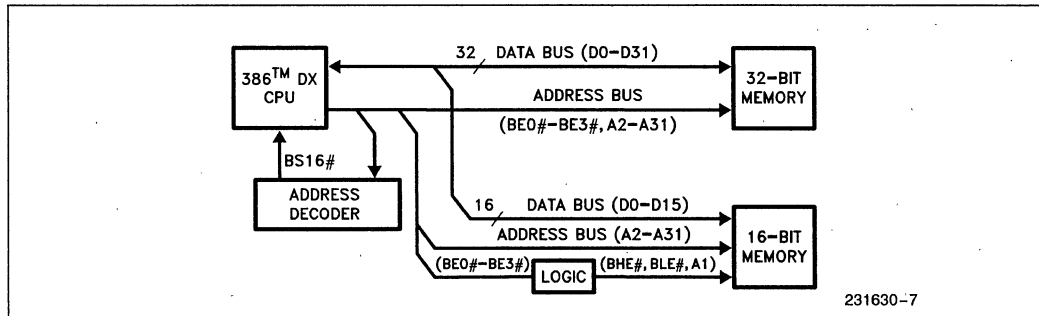


Figure 5-6. 386™ DX with 32-Bit and 16-Bit Memory

use pipelined address with 16-bit memories then BE0#–BE3# and W/R# are also decoded to determine when BS16# should be asserted. See **5.4.3.6 Pipelined Address with Dynamic Data Bus Sizing**.)

A2–A31 are directly usable for addressing 32-bit and 16-bit devices. To address 16-bit devices, A1 and two byte enable signals are also needed.

To generate an A1 signal and two Byte Enable signals for 16-bit access, BE0#–BE3# should be decoded as in Table 5-7. Note certain combinations of BE0#–BE3# are never generated by the 386 DX, leading to “don’t care” conditions in the decoder. Any BE0#–BE3# decoder, such as Figure 5-7, may use the non-occurring BE0#–BE3# combinations to its best advantage.

5.3.6 Operand Alignment

With the flexibility of memory addressing on the 386 DX, it is possible to transfer a logical operand that spans more than one physical Dword or word of memory or I/O. Examples are 32-bit Dwordoperands

beginning at addresses not evenly divisible by 4, or a 16-bit word operand split between two physical Dwords of the memory array.

Operand alignment and data bus size dictate when multiple bus cycles are required. Table 5-8 describes the transfer cycles generated for all combinations of logical operand lengths, alignment, and data bus sizing. When multiple bus cycles are required to transfer a multi-byte logical operand, the highest-order bytes are transferred first (but if BS16# asserted requires two 16-bit cycles be performed, that part of the transfer is low-order first).

5.4 BUS FUNCTIONAL DESCRIPTION

5.4.1 Introduction

The 386 DX has separate, parallel buses for data and address. The data bus is 32-bits in width, and bidirectional. The address bus provides a 32-bit value using 30 signals for the 30 upper-order address bits and 4 Byte Enable signals to directly indicate the active bytes. These buses are interpreted and controlled via several associated definition or control signals.

Table 5-7. Generating A1, BHE # and BLE # for Addressing 16-Bit Devices

386™ DX Signals				16-Bit Bus Signals			Comments
BE3 #	BE2 #	BE1 #	BE0 #	A1	BHE #	BLE # (A0)	
H*	H*	H*	H*	x	x	x	x—no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	x—not contiguous bytes
H*	L*	H*	L*	x	x	x	
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	H	H	x—not contiguous bytes x—not contiguous bytes x—not contiguous bytes
L*	H*	H*	L*	x	x	x	
L*	H*	L*	H*	x	x	x	
L*	H*	L*	L*	x	x	x	
L	L	H	H	H	L	L	x—not contiguous bytes
L*	L*	H*	L*	x	x	x	
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

BLE # asserted when D0–D7 of 16-bit bus is active.
 BHE # asserted when D8–D15 of 16-bit bus is active.
 A1 low for all even words; A1 high for all odd words.

Key:

- x = don’t care
- H = high voltage level
- L = low voltage level
- * = a non-occurring pattern of Byte Enables; either none are asserted, or the pattern has Byte Enables asserted for non-contiguous bytes

The definition of each bus cycle is given by three definition signals: M/IO#, W/R# and D/C#. At the same time, a valid address is present on the byte enable signals BE0#–BE3# and other address signals A2–A31. A status signal, ADS#, indicates when the 386 DX issues a new bus cycle definition and address.

Collectively, the address bus, data bus and all associated control signals are referred to simply as “the bus”.

When active, the bus performs one of the bus cycles below:

- 1) read from memory space
- 2) locked read from memory space
- 3) write to memory space
- 4) locked write to memory space
- 5) read from I/O space (or coprocessor)
- 6) write to I/O space (or coprocessor)
- 7) interrupt acknowledge
- 8) indicate halt, or indicate shutdown

Table 5-2 shows the encoding of the bus cycle definition signals for each bus cycle. See section 5.2.5 **Bus Cycle Definition**.

The data bus has a dynamic sizing feature supporting 32- and 16-bit bus size. Data bus size is indicated to the 386 DX using its Bus Size 16 (BS16#) input. All bus functions can be performed with either data bus size.

When the 386 DX bus is not performing one of the activities listed above, it is either Idle or in the Hold Acknowledge state, which may be detected by external circuitry. The idle state can be identified by the 386 DX giving no further assertions on its address strobe output (ADS#) since the beginning of its most recent bus cycle, and the most recent bus cycle has been terminated. The hold acknowledge state is identified by the 386 DX asserting its hold acknowledge (HLDA) output.

The shortest time unit of bus activity is a bus state. A bus state is one processor clock period (two CLK2 periods) in duration. A complete data transfer occurs during a bus cycle, composed of two or more bus states.

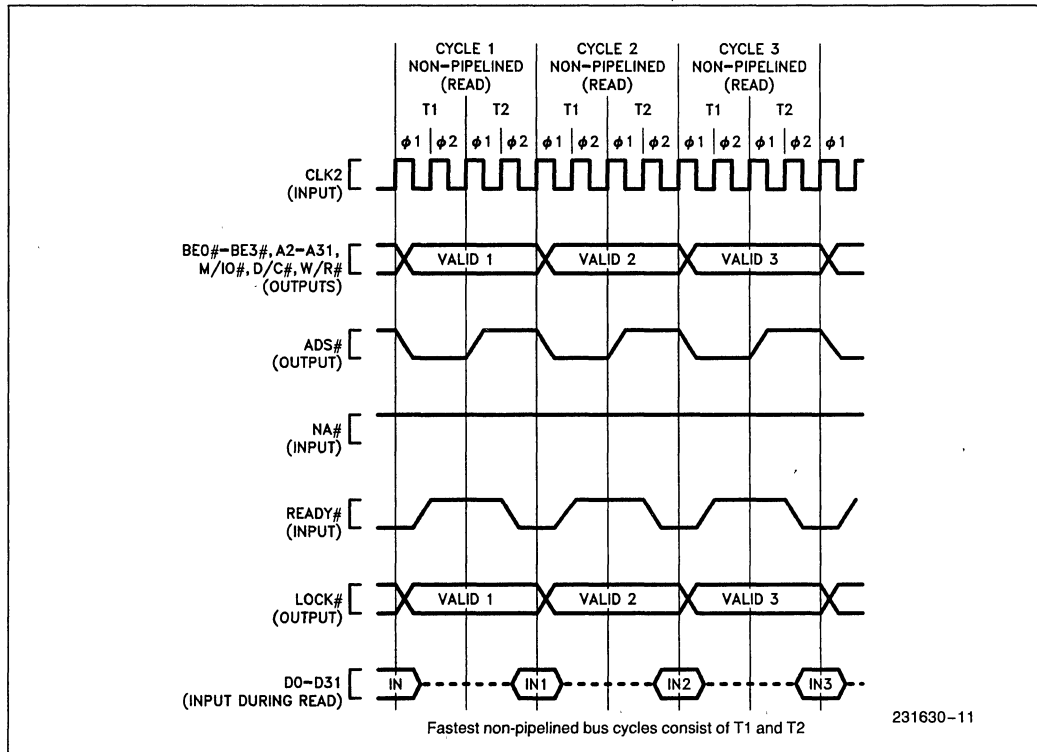


Figure 5-8. Fastest Read Cycles with Non-Pipelined Address Timing

The fastest 386 DX bus cycle requires only two bus states. For example, three consecutive bus read cycles, each consisting of two bus states, are shown by Figure 5-8. The bus states in each cycle are named **T1** and **T2**. Any memory or I/O address may be accessed by such a two-state bus cycle, if the external hardware is fast enough. The high-bandwidth, two-clock bus cycle realizes the full potential of fast main memory, or cache memory.

Every bus cycle continues until it is acknowledged by the external system hardware, using the 386 DX **READY#** input. Acknowledging the bus cycle at the end of the first **T2** results in the shortest bus cycle, requiring only **T1** and **T2**. If **READY#** is not immediately asserted, however, **T2** states are repeated indefinitely until the **READY#** input is sampled asserted.

5.4.2 Address Pipelining

The address pipelining option provides a choice of bus cycle timings. Pipelined or non-pipelined address timing is selectable on a cycle-by-cycle basis with the **Next Address (NA#)** input.

When address pipelining is not selected, the current address and bus cycle definition remain stable throughout the bus cycle.

When address pipelining is selected, the address (**BE0#–BE3#**, **A2–A31**) and definition (**W/R#**, **D/C#** and **M/I/O#**) of the next cycle are available before the end of the current cycle. To signal their availability, the 386 DX address status output (**ADS#**) is also asserted. Figure 5-9 illustrates the fastest read cycles with pipelined address timing.

Note from Figure 5-9 the fastest bus cycles using pipelined address require only two bus states, named **T1P** and **T2P**. Therefore cycles with pipelined address timing allow the same data bandwidth as non-pipelined cycles, but address-to-data access time is increased compared to that of a non-pipelined cycle.

By increasing the address-to-data access time, pipelined address timing reduces wait state requirements. For example, if one wait state is required with non-pipelined address timing, no wait states would be required with pipelined address.

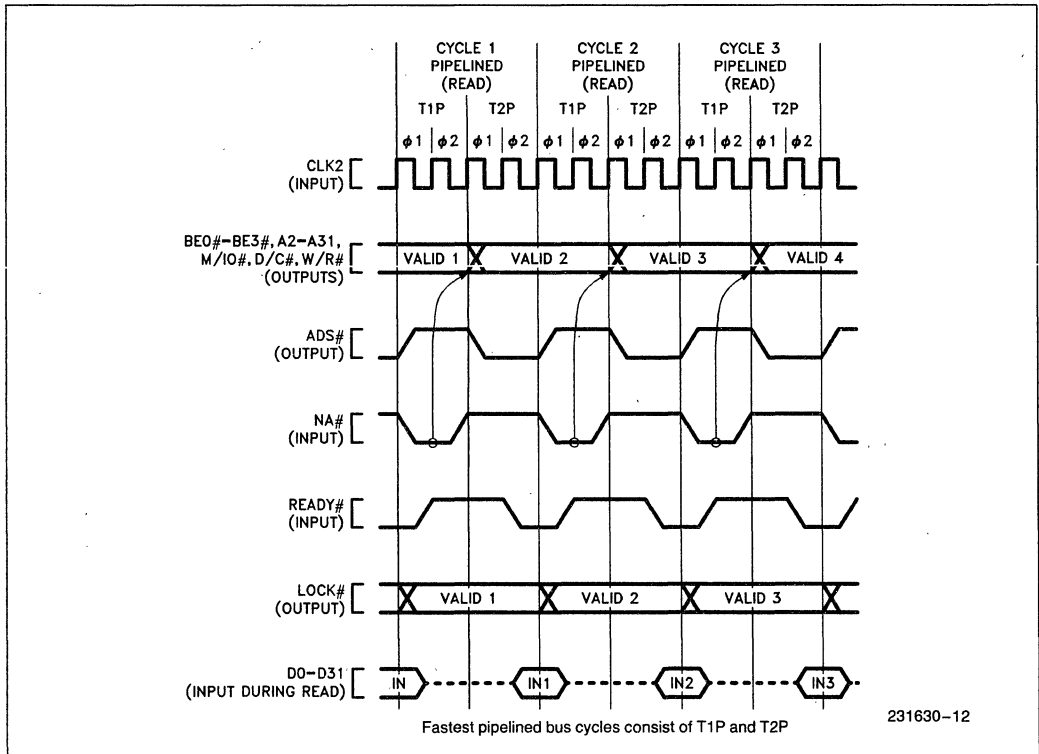


Figure 5-9. Fastest Read Cycles with Pipelined Address Timing

Pipelined address timing is useful in typical systems having address latches. In those systems, once an address has been latched, pipelined availability of the next address allows decoding circuitry to generate chip selects (and other necessary select signals) in advance, so selected devices are accessed immediately when the next cycle begins. In other words, the decode time for the next cycle can be overlapped with the end of the current cycle.

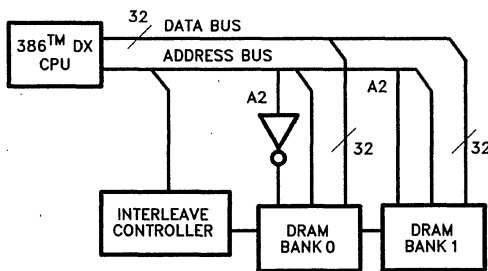
If a system contains a memory structure of two or more interleaved memory banks, pipelined address timing potentially allows even more overlap of activity. This is true when the interleaved memory controller is designed to allow the next memory operation

to begin in one memory bank while the current bus cycle is still activating another memory bank. Figure 5-10 shows the general structure of the 386 DX with 2-bank and 4-bank interleaved memory. Note each memory bank of the interleaved memory has full data bus width (32-bit data width typically, unless 16-bit bus size is selected).

Further details of pipelined address timing are given in 5.4.3.4 Pipelined Address, 5.4.3.5 Initiating and Maintaining Pipelined Address, 5.4.3.6 Pipelined Address with Dynamic Bus Sizing, and 5.4.3.7 Maximum Pipelined Address Usage with 16-Bit Bus Size.

TWO-BANK INTERLEAVED MEMORY

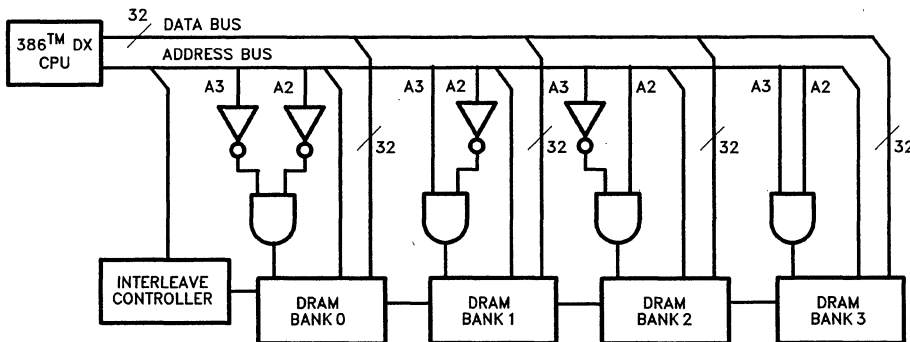
- a) Address signal A2 selects bank
- b) 32-bit datapath to each bank



231630-13

FOUR-BANK INTERLEAVED MEMORY

- a) Address signals A3 and A2 select bank
- b) 32-bit datapath to each bank



231630-14

Figure 5-10. 2-Bank and 4-Bank Interleaved Memory Structure

5.4.3 Read and Write Cycles

5.4.3.1 INTRODUCTION

Data transfers occur as a result of bus cycles, classified as read or write cycles. During read cycles, data is transferred from an external device to the processor. During write cycles data is transferred in the other direction, from the processor to an external device.

Two choices of address timing are dynamically selectable: non-pipelined, or pipelined. After a bus idle state, the processor always uses non-pipelined address timing. However, the NA# (Next Address) input may be asserted to select pipelined address timing for the next bus cycle. When pipelining is selected and the 386 DX has a bus request pending internally, the address and definition of the next cycle is made available even before the current bus cycle is acknowledged by READY#. Generally, the NA# input is sampled each bus cycle to select the desired address timing for the next bus cycle.

Two choices of physical data bus width are dynamically selectable: 32 bits, or 16 bits. Generally, the BS16# (Bus Size 16) input is sampled near the end of the bus cycle to confirm the physical data bus size applicable to the current cycle. Negation of BS16# indicates a 32-bit size, and assertion indicates a 16-bit bus size.

If 16-bit bus size is indicated, the 386 DX automatically responds as required to complete the transfer on a 16-bit data bus. Depending on the size and alignment of the operand, another 16-bit bus cycle may be required. Table 5-7 provides all details. When necessary, the 386 DX performs an additional 16-bit bus cycle, using D0–D15 in place of D16–D31.

Terminating a read cycle or write cycle, like any bus cycle, requires acknowledging the cycle by asserting the READY# input. Until acknowledged, the processor inserts wait states into the bus cycle, to allow adjustment for the speed of any external device. External hardware, which has decoded the address and bus cycle type asserts the READY# input at the appropriate time.

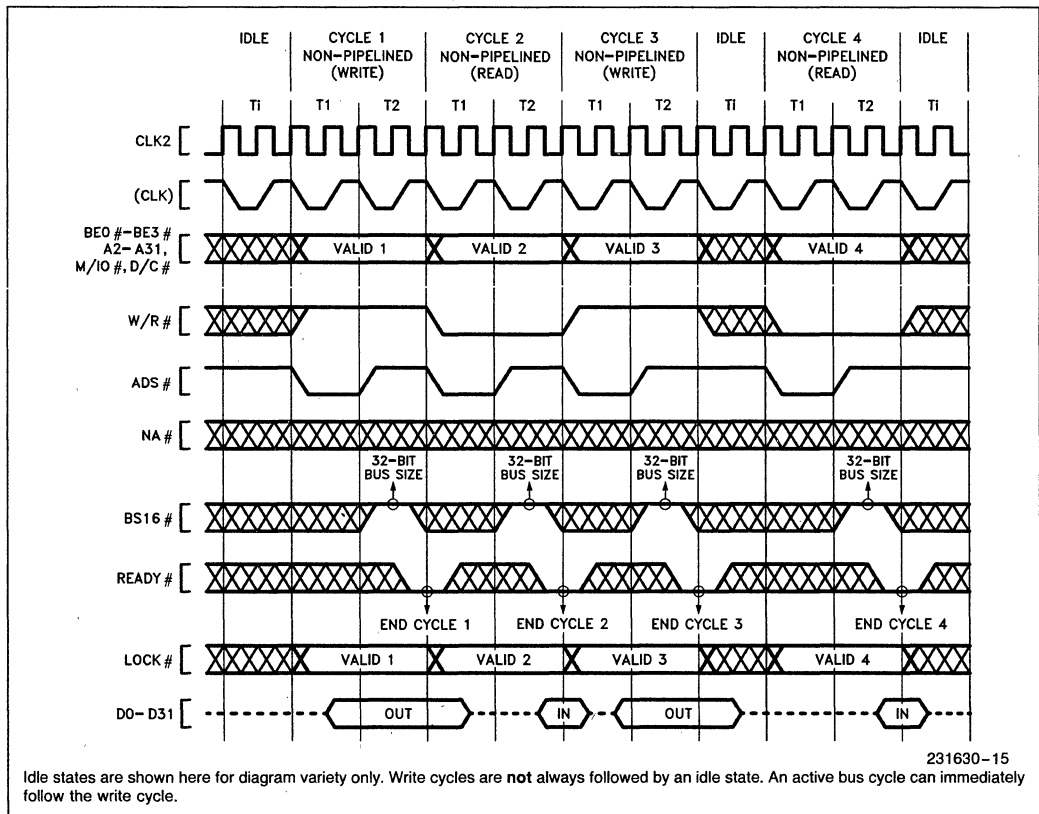


Figure 5-11. Various Bus Cycles and Idle States with Non-Pipelined Address (zero wait states)

At the end of the second bus state within the bus cycle, **READY#** is sampled. At that time, if external hardware acknowledges the bus cycle by asserting **READY#**, the bus cycle terminates as shown in Figure 5-11. If **READY#** is negated as in Figure 5-12, the cycle continues another bus state (a wait state) and **READY#** is sampled again at the end of that state. This continues indefinitely until the cycle is acknowledged by **READY#** asserted.

When the current cycle is acknowledged, the 386 DX terminates it. When a read cycle is acknowledged, the 386 DX latches the information present at its data pins. When a write cycle is acknowledged, the 386 DX write data remains valid throughout phase one of the next bus state, to provide write data hold time.

5.4.3.2 NON-PIPELINED ADDRESS

Any bus cycle may be performed with non-pipelined address timing. For example, Figure 5-11 shows a mixture of read and write cycles with non-pipelined address timing. Figure 5-11 shows the fastest possi-

ble cycles with non-pipelined address have two bus states per bus cycle. The states are named T1 and T2. In phase one of the T1, the address signals and bus cycle definition signals are driven valid, and to signal their availability, address status (**ADS#**) is simultaneously asserted.

During read or write cycles, the data bus behaves as follows. If the cycle is a read, the 386 DX floats its data signals to allow driving by the external device being addressed. **The 386 DX requires that all data bus pins be at a valid logic state (high or low) at the end of each read cycle, when **READY#** is asserted, even if all byte enables are not asserted. The system MUST be designed to meet this requirement.** If the cycle is a write, data signals are driven by the 386 DX beginning in phase two of T1 until phase one of the bus state following cycle acknowledgment.

Figure 5-12 illustrates non-pipelined bus cycles with one wait added to cycles 2 and 3. **READY#** is sampled negated at the end of the first T2 in cycles 2 and 3. Therefore cycles 2 and 3 have T2 repeated. At the end of the second T2, **READY#** is sampled asserted.

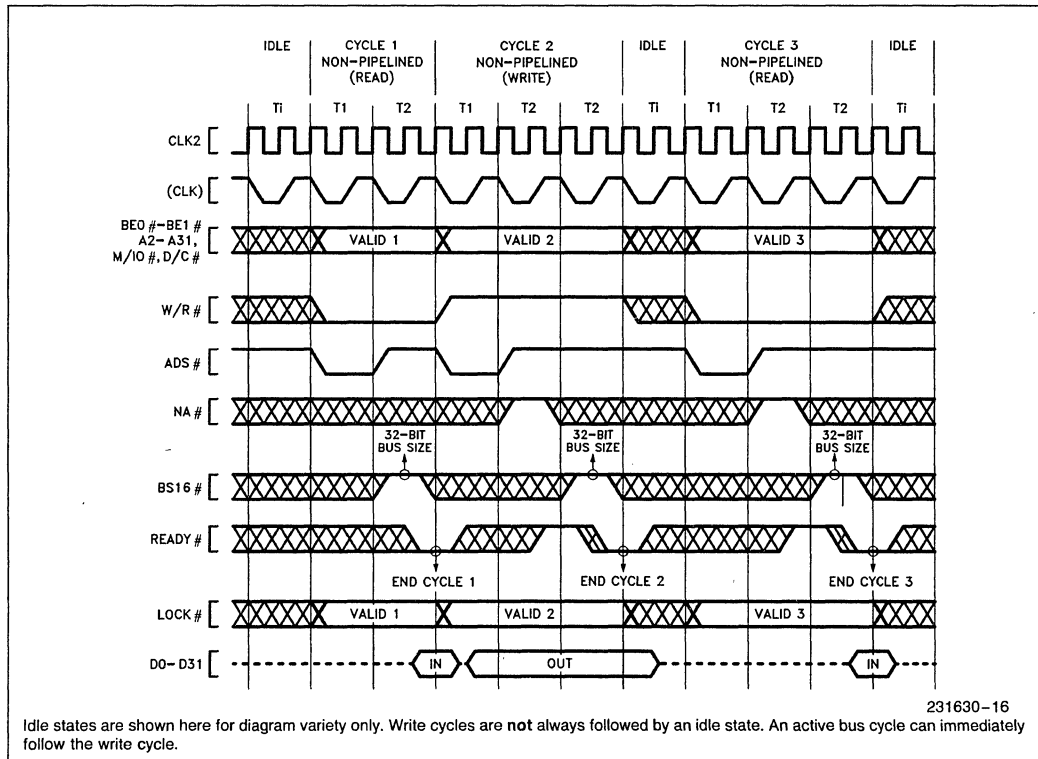


Figure 5-12. Various Bus Cycles and Idle States with Non-Pipelined Address (various number of wait states)

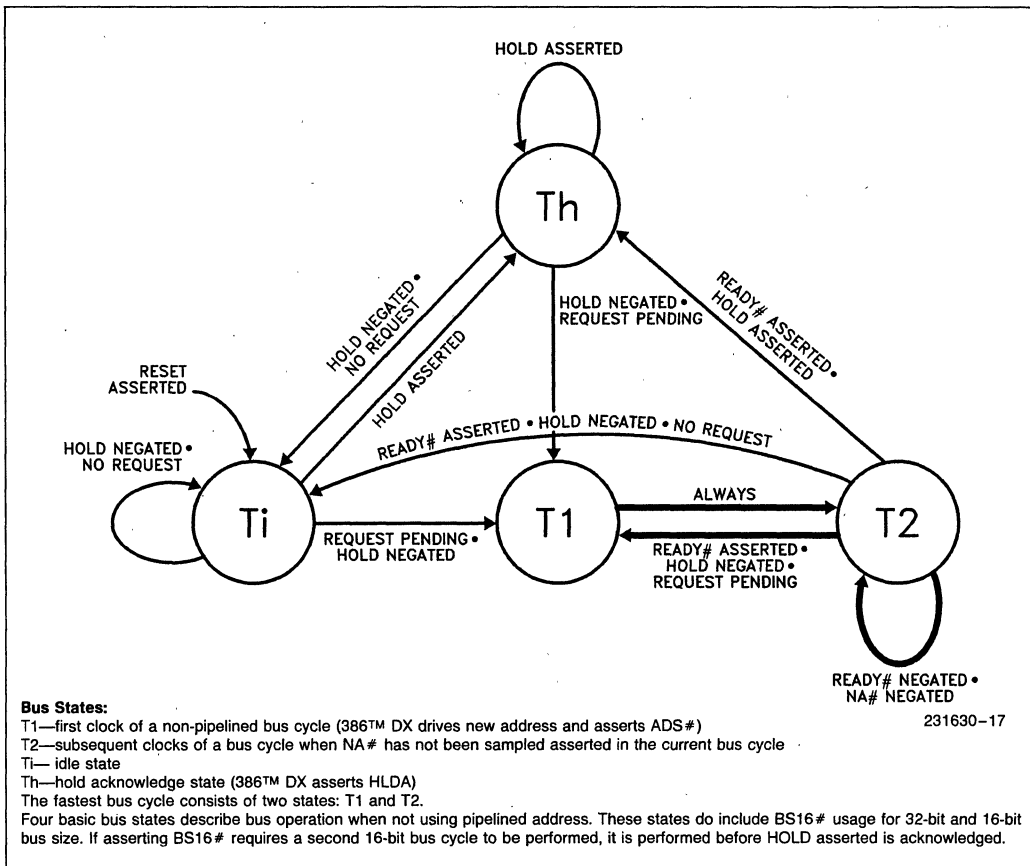


Figure 5-13. 386™ DX Bus States (not using pipelined address)

When address pipelining is not used, the address and bus cycle definition remain valid during all wait states. When wait states are added and you desire to maintain non-pipelined address timing, it is necessary to negate NA# during each T2 state except the last one, as shown in Figure 5-12 cycles 2 and 3. If NA# is sampled asserted during a T2 other than the last one, the next state would be T2I (for pipelined address) or T2P (for pipelined address) instead of another T2 (for non-pipelined address).

When address pipelining is not used, the bus states and transitions are completely illustrated by Figure 5-13. The bus transitions between four possible states: T1, T2, Ti, and Th. Bus cycles consist of T1 and T2, with T2 being repeated for wait states. Otherwise, the bus may be idle, in the Ti state, or in hold acknowledge, the Th state.

When address pipelining is not used, the bus state diagram is as shown in Figure 5-13. When the bus is

idle it is in state Ti. Bus cycles always begin with T1. T1 always leads to T2. If a bus cycle is not acknowledged during T2 and NA# is negated, T2 is repeated. When a cycle is acknowledged during T2, the following state will be T1 of the next bus cycle if a bus request is pending internally, or Ti if there is no bus request pending, or Th if the HOLD input is being asserted.

The bus state diagram in Figure 5-13 also applies to the use of BS16#. If the 386 DX makes internal adjustments for 16-bit bus size, the adjustments do not affect the external bus states. If an additional 16-bit bus cycle is required to complete a transfer on a 16-bit bus, it also follows the state transitions shown in Figure 5-13.

Use of pipelined address allows the 386 DX to enter three additional bus states not shown in Figure 5-13. Figure 5-20 in **5.4.3.4 Pipelined Address** is the complete bus state diagram, including pipelined address cycles.

5.4.3.3 NON-PIPELINED ADDRESS WITH DYNAMIC DATA BUS SIZING

The physical data bus width for any non-pipelined bus cycle can be either 32-bits or 16-bits. At the beginning of the bus cycle, the processor behaves as if the data bus is 32-bits wide. When the bus cycle is acknowledged, by asserting READY# at the end of a T2 state, the most recent sampling of BS16# determines the data bus size for the cycle being acknowledged. If BS16# was most recently negated, the physical data bus size is defined as

32 bits. If BS16# was most recently asserted, the size is defined as 16 bits.

When BS16# is asserted and two 16-bit bus cycles are required to complete the transfer, BS16# must be asserted during the second cycle; 16-bit bus size is not assumed. Like any bus cycle, the second 16-bit cycle must be acknowledged by asserting READY#.

When a second 16-bit bus cycle is required to complete the transfer over a 16-bit bus, the addresses

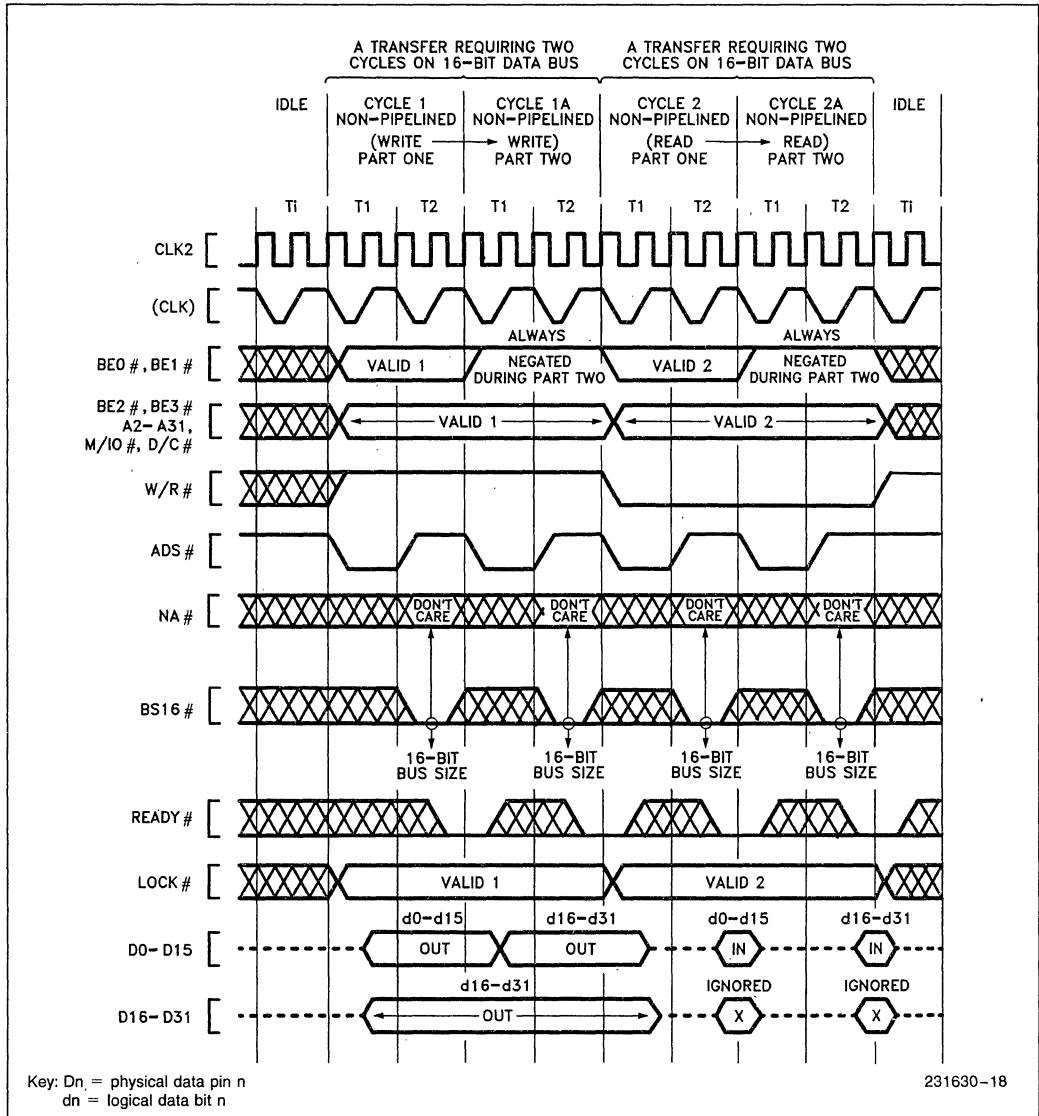


Figure 5-14. Asserting BS16# (zero wait states, non-pipelined address)

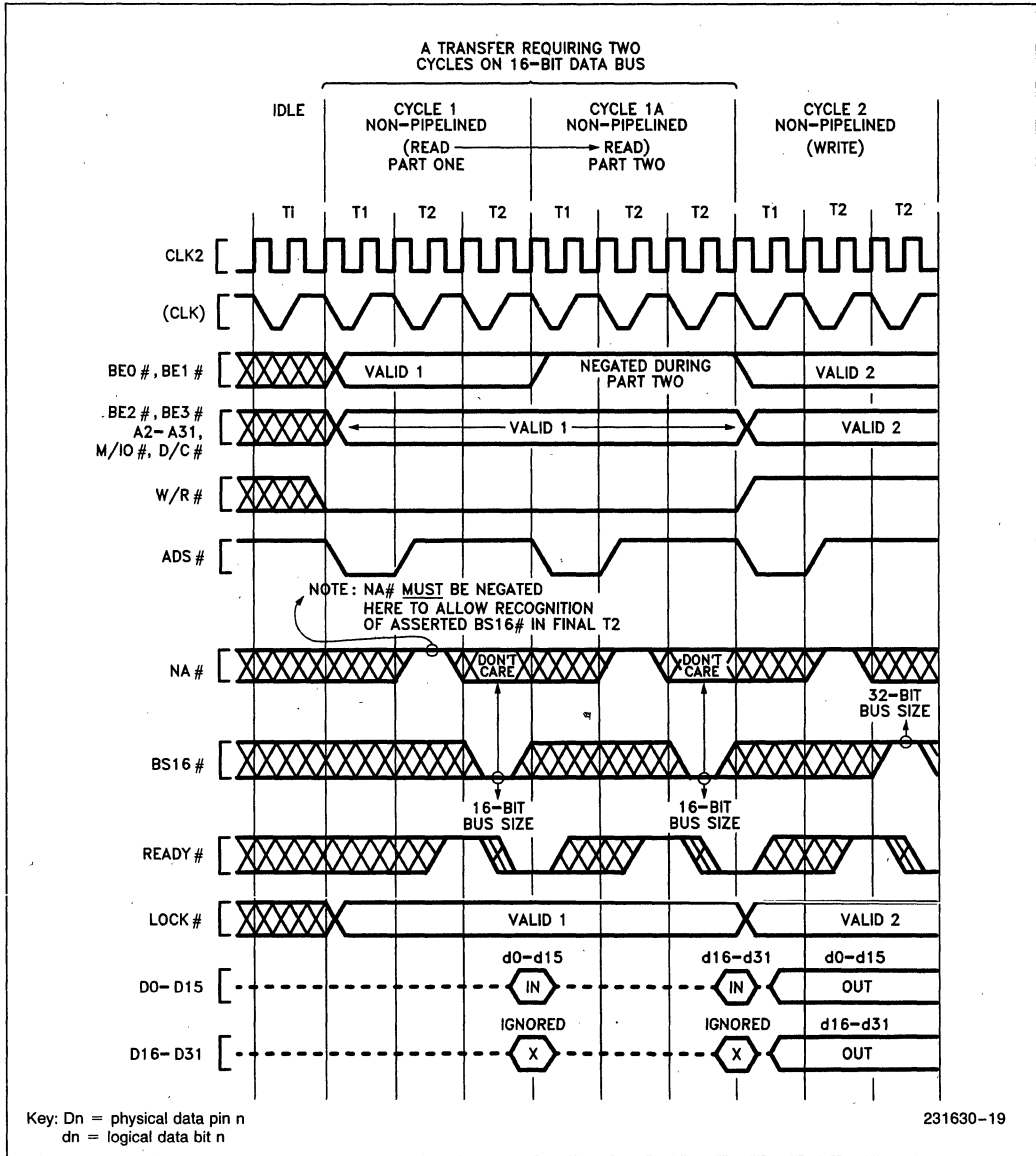


Figure 5-15. Asserting BS16# (one wait state, non-pipelined address)

generated for the two 16-bit bus cycles are closely related to each other. The addresses are the same except BE0# and BE1# are always negated for the second cycle. This is because data on D0-D15 was already transferred during the first 16-bit cycle.

Figures 5-14 and 5-15 show cases where assertion of BS16# requires a second 16-bit cycle for complete operand transfer. Figure 5-14 illustrates cycles

without wait states. Figure 5-15 illustrates cycles with one wait state. In Figure 5-15 cycle 1, the bus cycle during which BS16# is asserted, note that NA# must be negated in the T2 state(s) prior to the last T2 state. This is to allow the recognition of BS16# asserted in the final T2 state. The relation of NA# and BS16# is given fully in **5.4.3.4 Pipelined Address**, but Figure 5-15 illustrates this only precaution you need to know when using BS16# with non-pipelined address.

5.4.3.4 PIPELINED ADDRESS

Address pipelining is the option of requesting the address and the bus cycle definition of the next, internally pending bus cycle before the current bus cycle is acknowledged with **READY#** asserted. **ADS#** is asserted by the 386 DX when the next address is issued. The address pipelining option is controlled on a cycle-by-cycle basis with the **NA#** input signal.

Once a bus cycle is in progress and the current address has been valid for at least one entire bus state, the **NA#** input is sampled at the end of every phase one until the bus cycle is acknowledged. During non-pipelined bus cycles, therefore, **NA#** is sampled at the end of phase one in every T2. An example is Cycle 2 in Figure 5-16, during which **NA#** is sampled at the end of phase one of every T2 (it was asserted once during the first T2 and has no further effect during that bus cycle).

If **NA#** is sampled asserted, the 386 DX is free to drive the address and bus cycle definition of the next bus cycle, and assert **ADS#**, as soon as it has a bus request internally pending. It may drive the next address as early as the next bus state, whether the current bus cycle is acknowledged at that time or not.

Regarding the details of address pipelining, the 386 DX has the following characteristics:

- 1) For **NA#** to be sampled asserted, **BS16#** must be negated at that sampling window (see Figure 5-16 Cycles 2 through 4, and Figure 5-17 Cycles 1 through 4). If **NA#** and **BS16#** are both sampled asserted during the last T2 period of a bus cycle, **BS16#** asserted has priority. Therefore, if both are asserted, the current bus size is taken to be 16 bits and the next address is not pipelined.

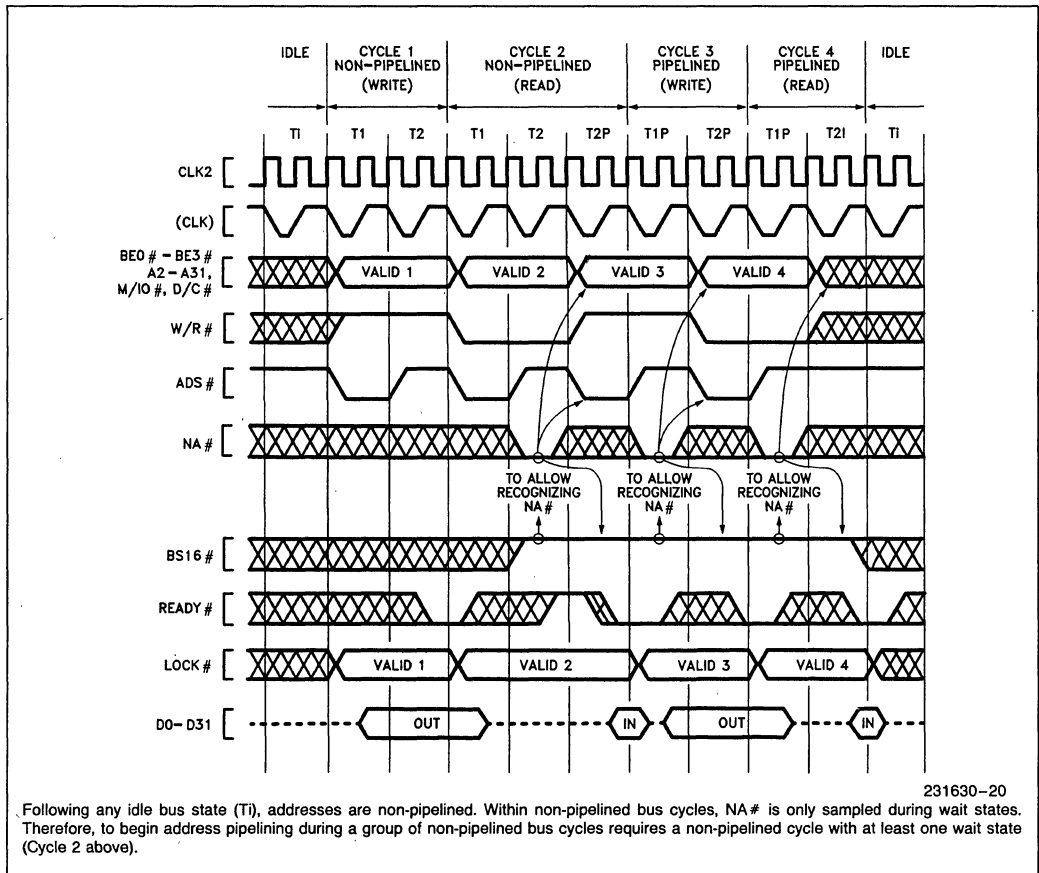


Figure 5-16. Transitioning to Pipelined Address During Burst of Bus Cycles

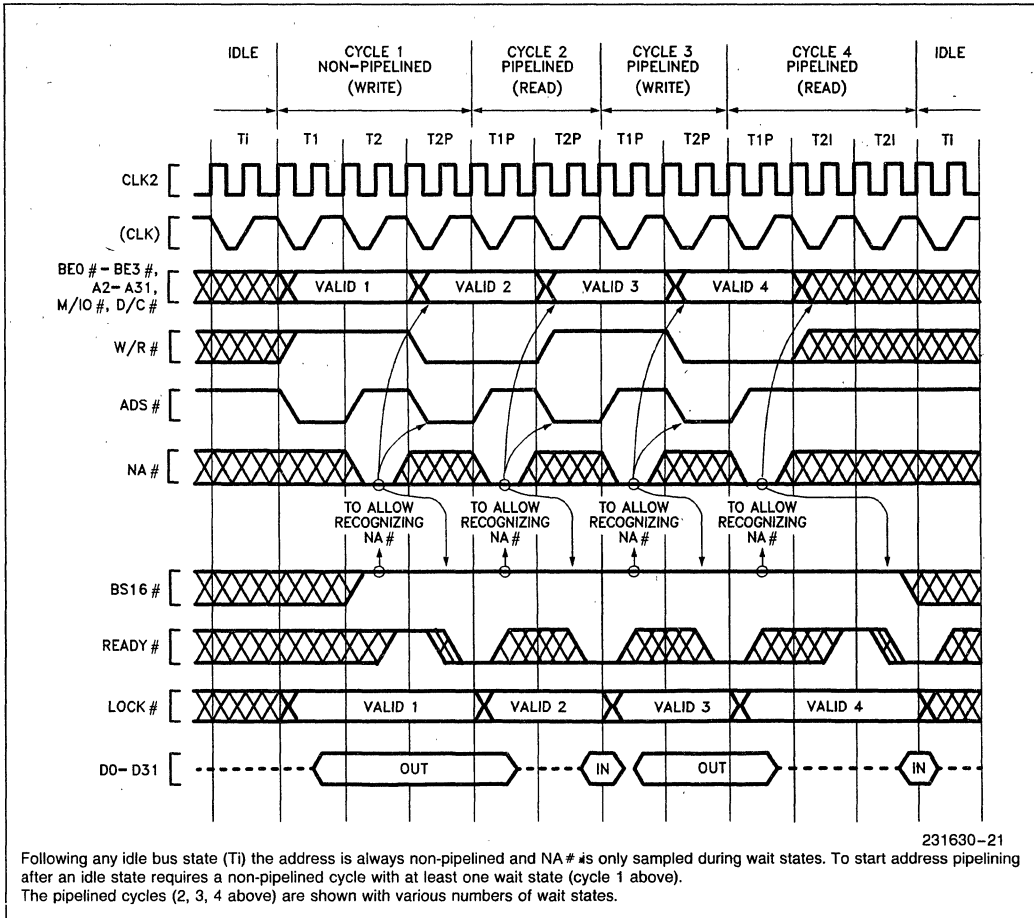


Figure 5-17. Fastest Transition to Pipelined Address Following Idle Bus State

- 2) The next address may appear as early as the bus state after NA# was sampled asserted (see Figures 5-16 or 5-17). In that case, state T2P is entered immediately. However, when there is not an internal bus request already pending, the next address will not be available immediately after NA# is asserted and T2I is entered instead of T2P (see Figure 5-19 Cycle 3). Provided the current bus cycle isn't yet acknowledged by READY# asserted, T2P will be entered as soon as the 386 DX does drive the next address. External hardware should therefore observe the ADS# output as confirmation the next address is actually being driven on the bus.
- 3) Once NA# is sampled asserted, the 386 DX commits itself to the highest priority bus request that is pending internally. It can no longer perform another 16-bit transfer to the same address should BS16# be asserted externally, so thereafter

must assume the current bus size is 32 bits. Therefore if NA# is sampled asserted within a bus cycle, BS16# must be negated thereafter in that bus cycle (see Figures 5-16, 5-17, 5-19). Consequently, do not assert NA# during bus cycles which must have BS16# driven asserted. See 5.4.3.6 Dynamic Bus Sizing with Pipelined Address.

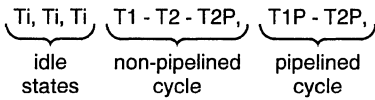
- 4) Any address which is validated by a pulse on the 386 DX ADS# output will remain stable on the address pins for at least two processor clock periods. The 386 DX cannot produce a new address more frequently than every two processor clock periods (see Figures 5-16, 5-17, 5-19).
- 5) Only the address and bus cycle definition of the very next bus cycle is available. The pipelining capability cannot look further than one bus cycle ahead (see Figure 5-19 Cycle 1).

The complete bus state transition diagram, including operation with pipelined address is given by 5-20. Note it is a superset of the diagram for non-pipelined address only, and the three additional bus states for pipelined address are drawn in bold.

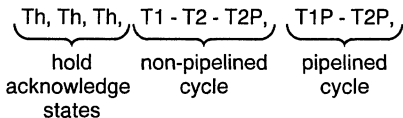
The fastest bus cycle with pipelined address consists of just two bus states, T1P and T2P (recall for non-pipelined address it is T1 and T2). T1P is the first bus state of a pipelined cycle.

5.4.3.5 INITIATING AND MAINTAINING PIPELINED ADDRESS

Using the state diagram Figure 5-20, observe the transitions from an idle state, Ti, to the beginning of a pipelined bus cycle, T1P. From an idle state Ti, the first bus cycle must begin with T1, and is therefore a non-pipelined bus cycle. The next bus cycle will be pipelined, however, provided NA# is asserted and the first bus cycle ends in a T2P state (the address for the next bus cycle is driven during T2P). The fastest path from an idle state to a bus cycle with pipelined address is shown in bold below:



T1-T2-T2P are the states of the bus cycle that establishes address pipelining for the next bus cycle, which begins with T1P. The same is true after a bus hold state, shown below:



The transition to pipelined address is shown functionally by Figure 5-17 Cycle 1. Note that Cycle 1 is used to transition into pipelined address timing for the subsequent Cycles 2, 3 and 4, which are pipelined. The NA# input is asserted at the appropriate time to select address pipelining for Cycles 2, 3 and 4.

Once a bus cycle is in progress and the current address has become valid, the NA# input is sampled at the end of every phase one, beginning with the next bus state, until the bus cycle is acknowledged. During Figure 5-17 Cycle 1 therefore, sampling begins in T2. Once NA# is sampled asserted during the current cycle, the 386 DX is free to drive a new address and bus cycle definition on the bus as early as the next bus state. In Figure 5-16 Cycle 1 for example, the next address is driven during state T2P. Thus Cycle 1 makes the transition to pipelined address timing, since it begins with T1 but ends with T2P. Because the address for Cycle 2 is available before Cycle 2 begins, Cycle 2 is called a pipelined bus cycle, and it begins with T1P. Cycle 2 begins as soon as READY# asserted terminates Cycle 1.

Example transition bus cycles are Figure 5-17 Cycle 1 and Figure 5-16 Cycle 2. Figure 5-17 shows transition during the very first cycle after an idle bus state, which is the fastest possible transition into address pipelining. Figure 5-16 Cycle 2 shows a transition cycle occurring during a burst of bus cycles. In any case, a transition cycle is the same whenever it occurs: it consists at least of T1, T2 (you assert NA# at that time), and T2P (provided the 386 DX has an internal bus request already pending, which it almost always has). T2P states are repeated if wait states are added to the cycle.

Note three states (T1, T2 and T2P) are only required in a bus cycle performing a **transition** from non-pipelined address into pipelined address timing, for example Figure 5-17 Cycle 1. Figure 5-17 Cycles 2, 3 and 4 show that address pipelining can be maintained with two-state bus cycles consisting only of T1P and T2P.

Once a pipelined bus cycle is in progress, pipelined timing is maintained for the next cycle by asserting NA# and detecting that the 386 DX enters T2P during the current bus cycle. The current bus cycle must end in state T2P for pipelining to be maintained in the next cycle. T2P is identified by the assertion of ADS#. Figures 5-16 and 5-17 however, each show pipelining ending after Cycle 4 because Cycle 4 ends in T2I. This indicates the 386 DX didn't have an internal bus request prior to the acknowledgement of Cycle 4. If a cycle ends with a T2 or T2I, the next cycle will not be pipelined.

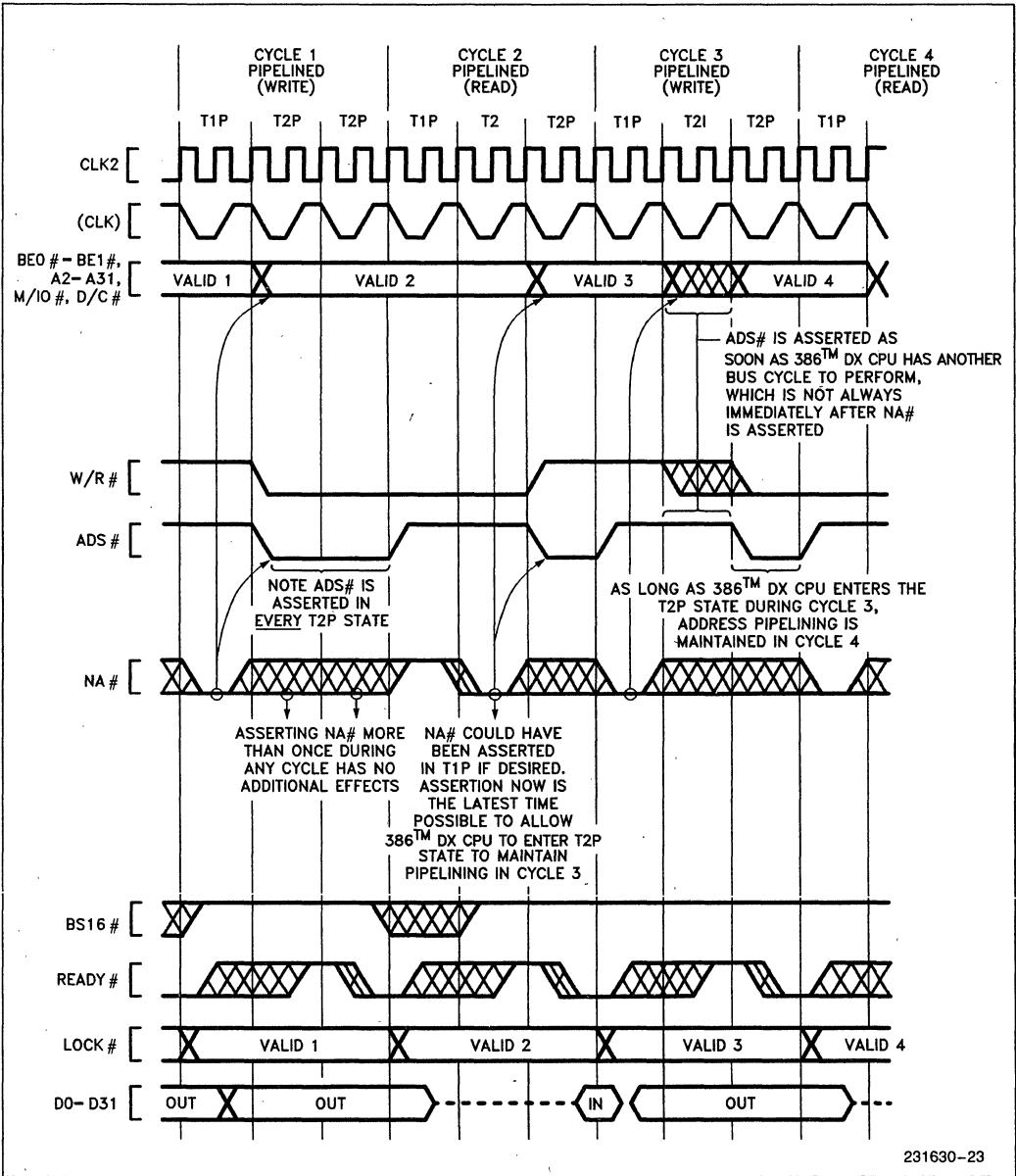


Figure 5-19. Details of Address Pipelining During Cycles with Wait States

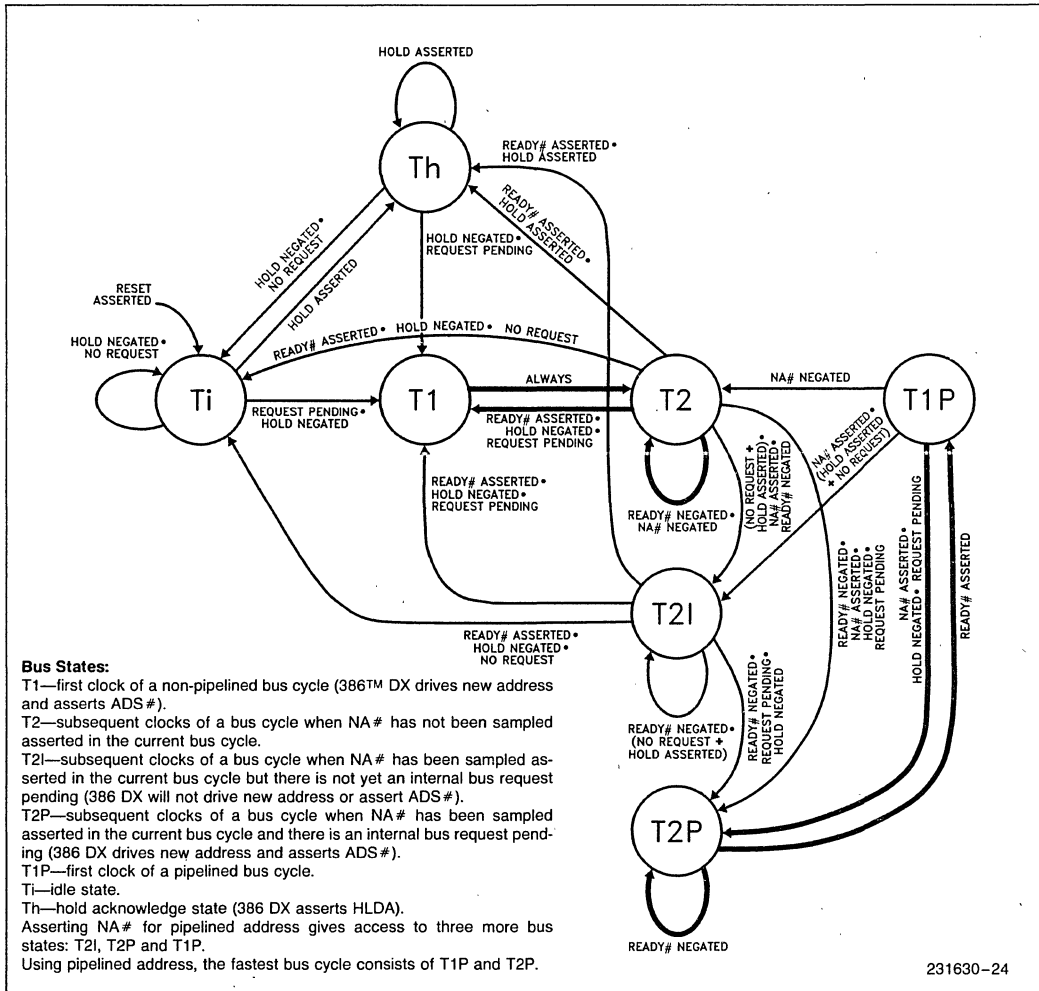


Figure 5-20. 386™ DX Complete Bus States (including pipelined address)

Realistically, address pipelining is almost always maintained as long as NA# is sampled asserted. This is so because in the absence of any other request, a code prefetch request is always internally pending until the instruction decoder and code prefetch queue are completely full. Therefore address pipelining is maintained for long bursts of bus cycles, if the bus is available (i.e., HOLD negated) and NA# is sampled asserted in each of the bus cycles.

5.4.3.6 PIPELINED ADDRESS WITH DYNAMIC DATA BUS SIZING

The BS16# feature allows easy interface to 16-bit data buses. When asserted, the 386 DX bus

interface hardware performs appropriate action to make the transfer using a 16-bit data bus connected on D0–D15.

There is a degree of interaction, however, between the use of Address Pipelining and the use of Bus Size 16. The interaction results from the multiple bus cycles required when transferring 32-bit operands over a 16-bit bus. If the operand requires both 16-bit halves of the 32-bit bus, the appropriate 386 DX action is a second bus cycle to complete the operand's transfer. It is this necessity that conflicts with NA# usage.

When NA# is sampled asserted, the 386 DX commits itself to perform the next inter-

nally pending bus request, and is allowed to drive the next internally pending address onto the bus. Asserting NA# therefore makes it impossible for the next bus cycle to again access the current address on A2-A31, such as may be required when BS16# is asserted by the external hardware.

To avoid conflict, the 386 DX is designed with following two provisions:

1) To avoid conflict, BS16# must be negated in the current bus cycle if NA# has already been

sampled asserted in the current cycle. If NA# is sampled asserted, the current data bus size is assumed to be 32 bits.

2) To also avoid conflict, if NA# and BS16# are both asserted during the same sampling window, BS16# asserted has priority and the 386 DX acts as if NA# was negated at that time. Internal 386 DX circuitry, shown conceptually in Figure 5-18, assures that BS16# is sampled asserted and NA# is sampled negated if both inputs are externally asserted at the same sampling window.

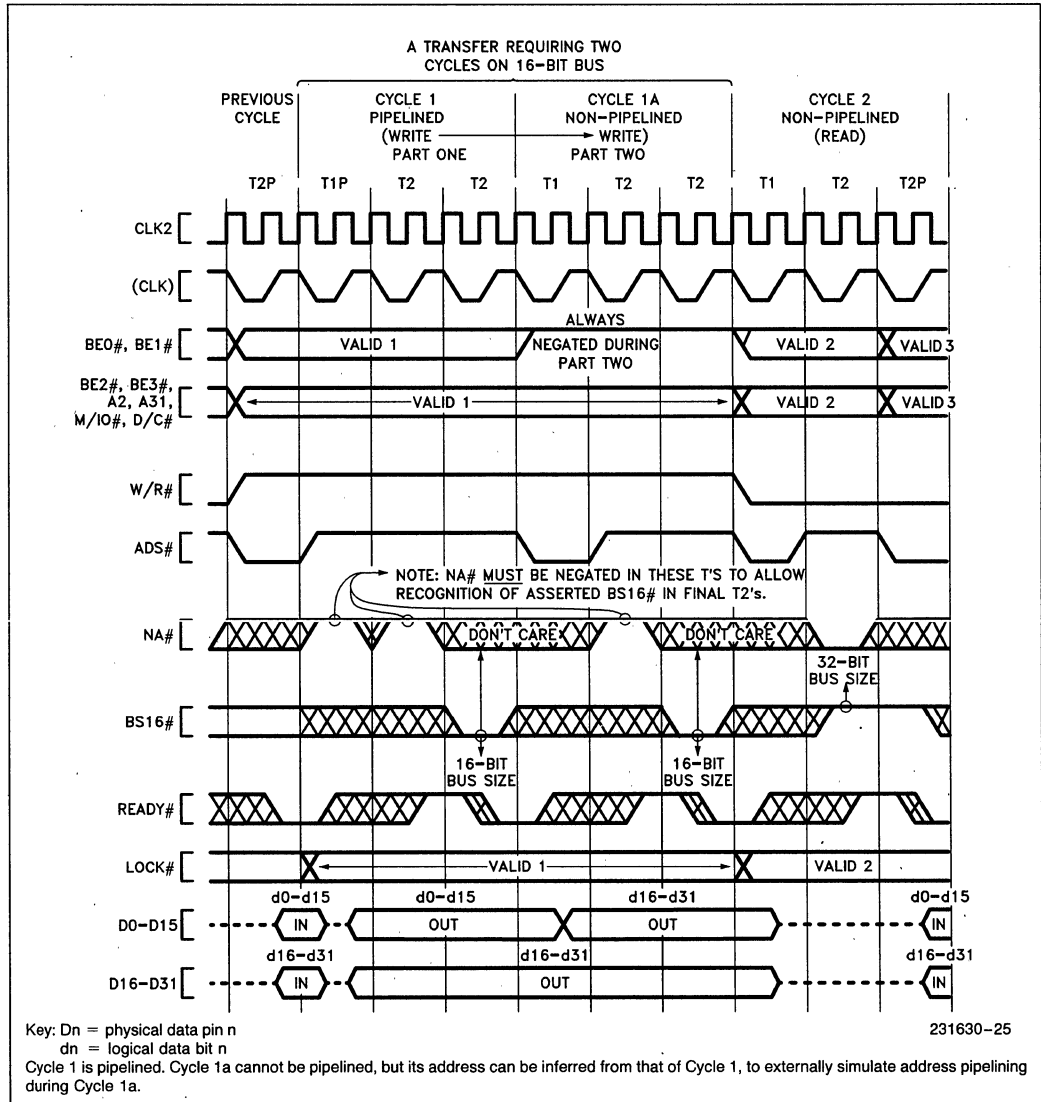


Figure 5-21. Using NA# and BS16#

Certain types of 16-bit or 8-bit operands require no adjustment for correct transfer on a 16-bit bus. Those are read or write operands using only the lower half of the data bus, and write operands using only the upper half of the bus since the 386 DX simultaneously duplicates the write data on the lower half of the data bus. For these patterns of Byte Enables and the R/W# signals, BS16# need not be asserted at the 386 DX allowing NA# to be asserted during the bus cycle if desired.

forms two interrupt acknowledge cycles. These bus cycles are similar to read cycles in that bus definition signals define the type of bus activity taking place, and each cycle continues until acknowledged by READY# sampled asserted.

The state of A2 distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A31–A3 low, A2 high, BE3#–BE1# high, and BE0# low). The address driven during the second interrupt acknowledge cycle is 0 (A31–A2 low, BE3#–BE1# high, BE0# low).

5.4.4 Interrupt Acknowledge (INTA) Cycles

In response to an interrupt request on the INTR input when interrupts are enabled, the 386 DX per-

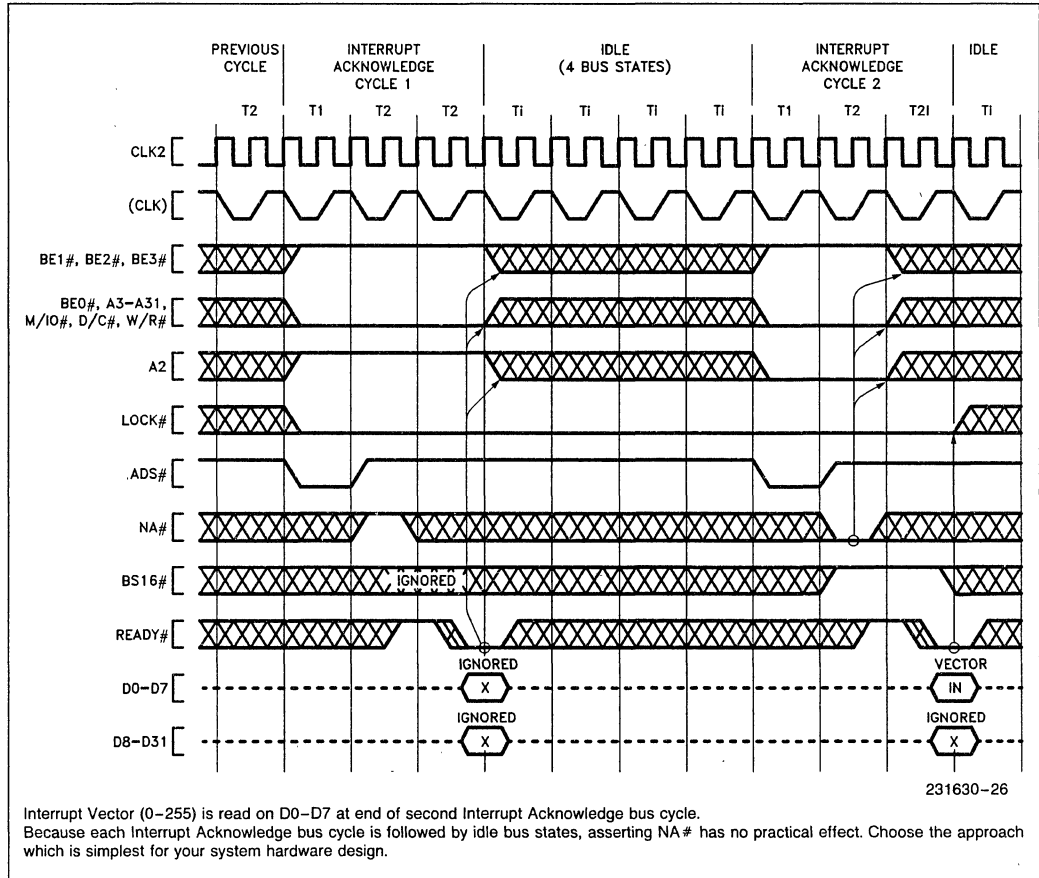


Figure 5-22. Interrupt Acknowledge Cycles

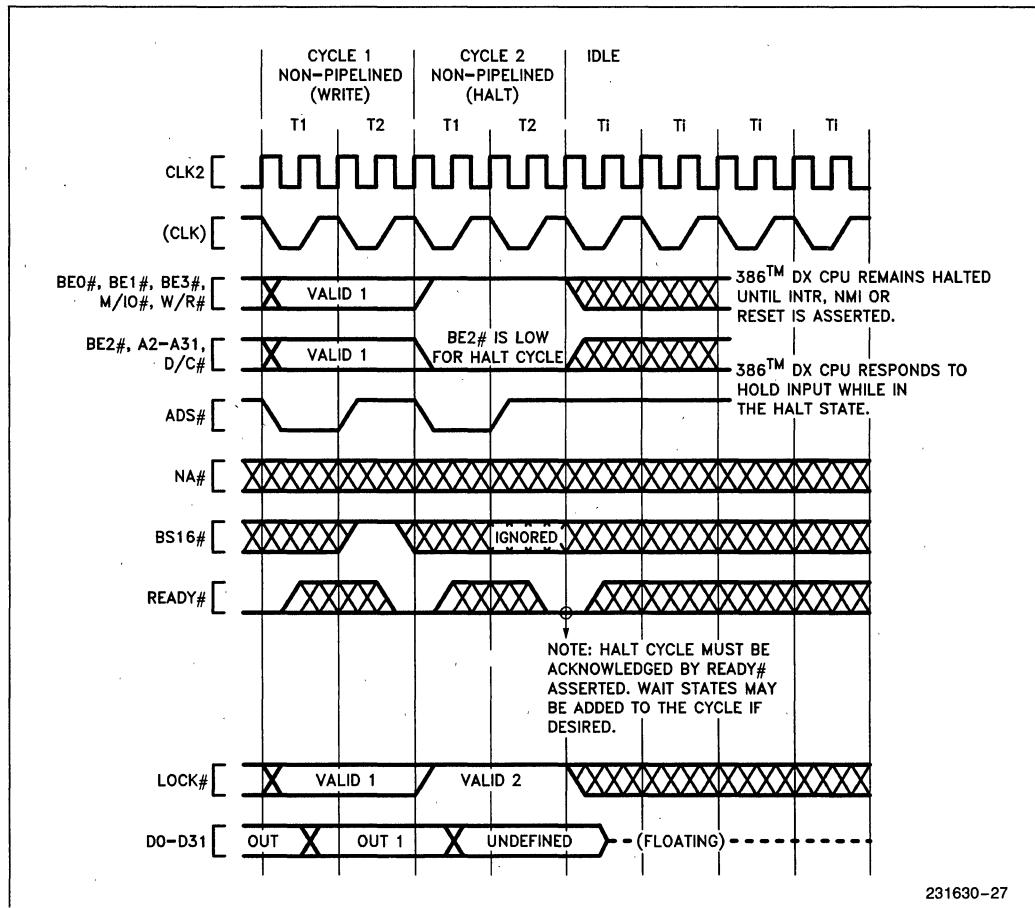


Figure 5-23. Halt Indication Cycle

The LOCK# output is asserted from the beginning of the first interrupt acknowledge cycle until the end of the second interrupt acknowledge cycle. Four idle bus states, T_i, are inserted by the 386 DX between the two interrupt acknowledge cycles, allowing for compatibility with spec TRHRL of the 8259A Interrupt Controller.

During both interrupt acknowledge cycles, D0-D31 float. No data is read at the end of the first interrupt acknowledge cycle. At the end of the second interrupt acknowledge cycle, the 386 DX will read an external interrupt vector from D0-D7 of the data bus. The vector indicates the specific interrupt number (from 0-255) requiring service.

5.4.5 Halt Indication Cycle

The 386 DX halts as a result of executing a HALT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the state of the bus definition signals shown in 5.2.5 Bus Cycle Definition and a byte address of 2. BE0# and BE2# are the only signals distinguishing halt indication from shut-down indication, which drives an address of 0. During the halt cycle undefined data is driven on D0-D31. The halt indication cycle must be acknowledged by READY# asserted.

A halted 386 DX resumes execution when INTR (if interrupts are enabled) or NMI or RESET is asserted.

5.4.6 Shutdown Indication Cycle

The 386 DX shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the state of the bus definition signals shown in 5.2.5 Bus Cycle Definition and a byte address of 0. BE0# and BE2#

are the only signals distinguishing shutdown indication from halt indication, which drives an address of 2. During the shutdown cycle undefined data is driven on D0–D31. The shutdown indication cycle must be acknowledged by READY# asserted.

A shutdown 386 DX resumes execution when NMI or RESET is asserted.

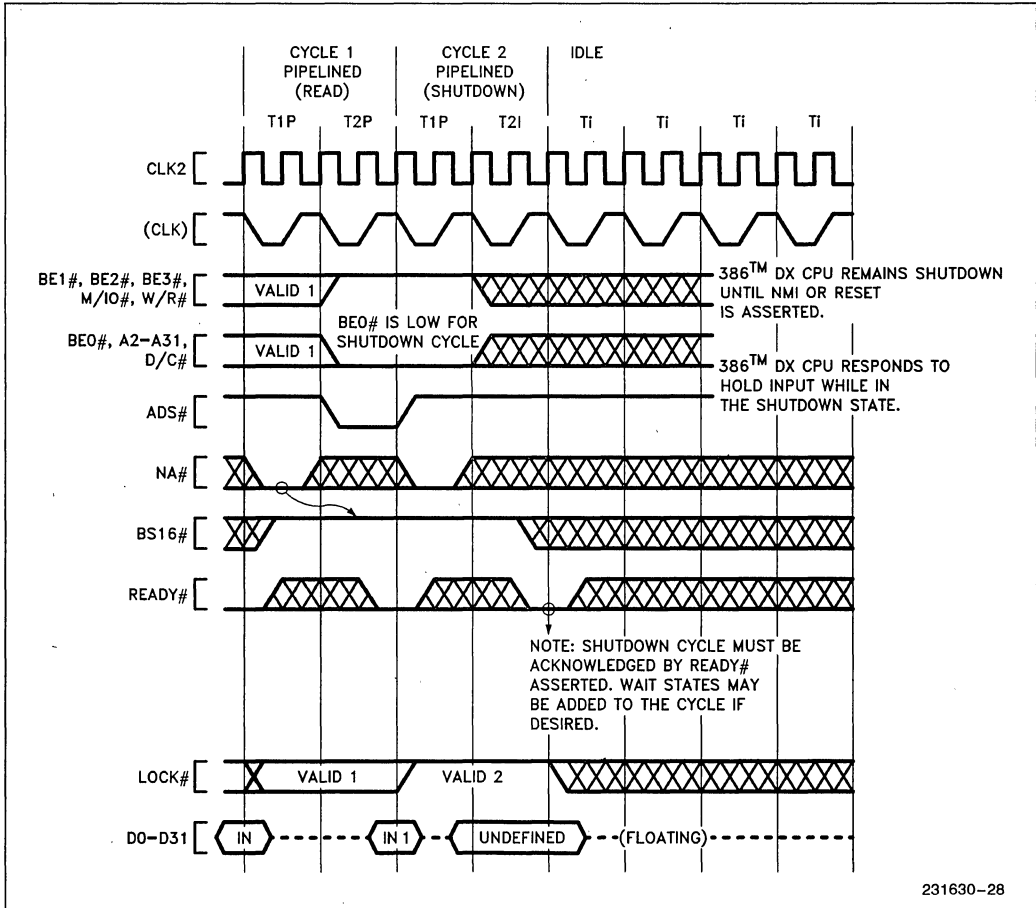


Figure 5-24. Shutdown Indication Cycle

5.5 OTHER FUNCTIONAL DESCRIPTIONS

5.5.1 Entering and Exiting Hold Acknowledge

The bus hold acknowledge state, Th, is entered in response to the HOLD input being asserted. In the bus hold acknowledge state, the 386 DX floats all output or bidirectional signals, except for HLDA. HLDA is asserted as long as the 386 DX remains in the bus hold acknowledge state. In the bus hold acknowledge state, all inputs except HOLD, RESET, BUSY#, ERROR#, and PEREQ are ignored (also up to one rising edge on NMI is remembered for processing when HOLD is no longer asserted).

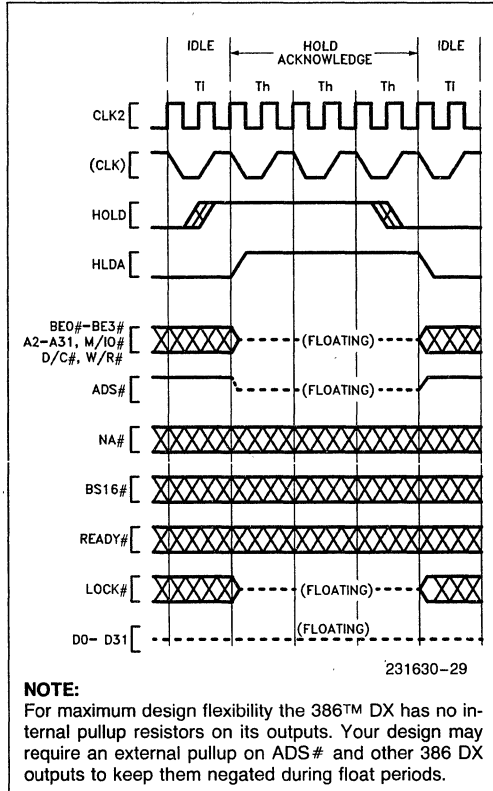


Figure 5-25. Requesting Hold from Idle Bus

Th may be entered from a bus idle state as in Figure 5-25 or after the acknowledgement of the current physical bus cycle if the LOCK# signal is not asserted, as in Figures 5-26 and 5-27. If HOLD is asserted during a locked bus cycle, the 386 DX may execute one unlocked bus cycle before acknowledging HOLD. If asserting BS16# requires a second 16-bit

bus cycle to complete a physical operand transfer, it is performed before HOLD is acknowledged, although the bus state diagrams in Figures 5-13 and 5-20 do not indicate that detail.

Th is exited in response to the HOLD input being negated. The following state will be Ti as in Figure 5-25 if no bus request is pending. The following bus state will be T1 if a bus request is internally pending, as in Figures 5-26 and 5-27.

Th is also exited in response to RESET being asserted.

If a rising edge occurs on the edge-triggered NMI input while in Th, the event is remembered as a non-maskable interrupt 2 and is serviced when Th is exited, unless of course, the 386 DX is reset before Th is exited.

5.5.2 Reset During Hold Acknowledge

RESET being asserted takes priority over HOLD being asserted. Therefore, Th is exited in response to the RESET input being asserted. If RESET is asserted while HOLD remains asserted, the 386 DX drives its pins to defined states during reset, as in Table 5-3 Pin State During Reset, and performs internal reset activity as usual.

If HOLD remains asserted when RESET is negated, the 386 DX enters the hold acknowledge state before performing its first bus cycle, provided HOLD is still asserted when the 386 DX would otherwise perform its first bus cycle. If HOLD remains asserted when RESET is negated, the BUSY# input is still sampled as usual to determine whether a self test is being requested, and ERROR# is still sampled as usual to determine whether a 387 DX coprocessor vs. an 80287 (or none) is present.

5.5.3 Bus Activity During and Following Reset

RESET is the highest priority input signal, capable of interrupting any processor activity when it is asserted. A bus cycle in progress can be aborted at any stage, or idle states or bus hold acknowledge states discontinued so that the reset state is established.

RESET should remain asserted for at least 15 CLK2 periods to ensure it is recognized throughout the 386 DX, and at least 80 CLK2 periods if 386 DX self-test is going to be requested at the falling edge. RESET asserted pulses less than 15 CLK2 periods may not be recognized. RESET pulses less than 80 CLK2 periods followed by a self-test may cause the self-test to report a failure when no true failure exists.

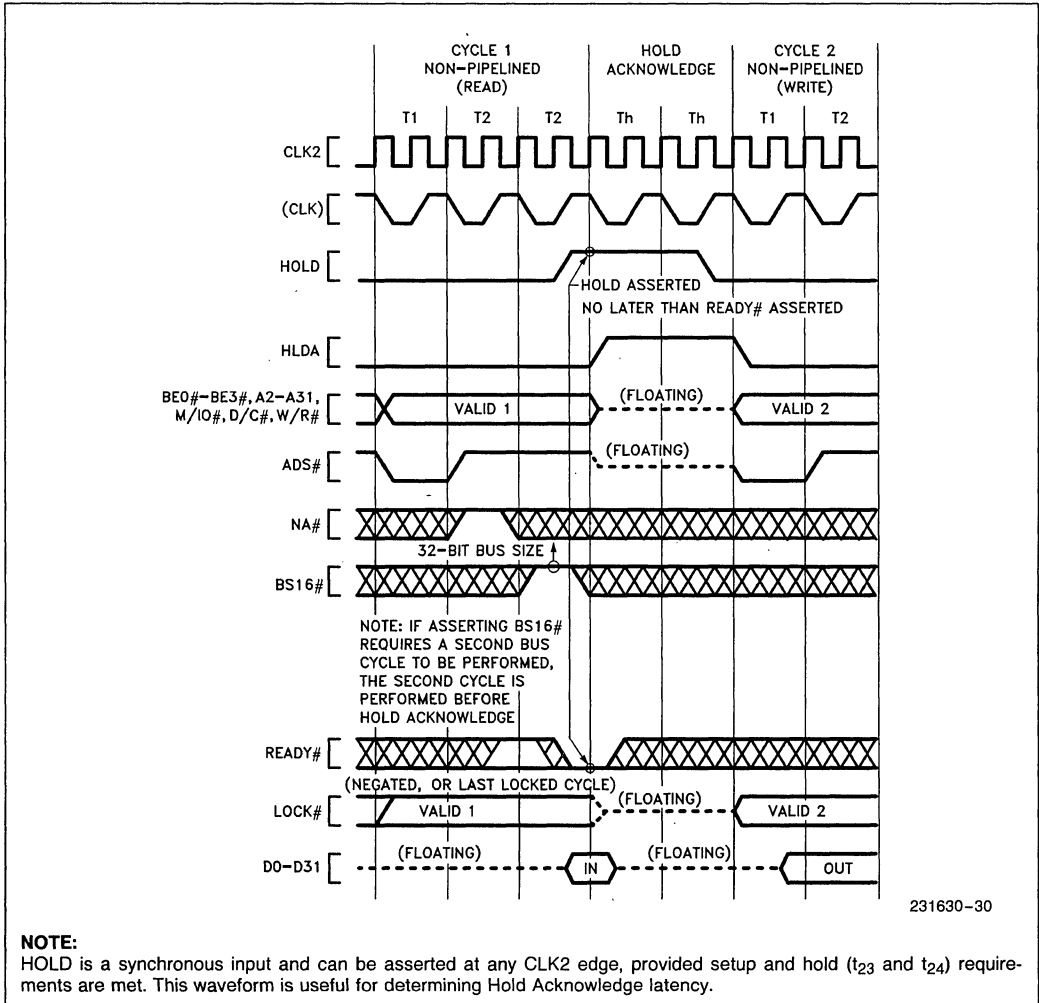


Figure 5-26. Requesting Hold from Active Bus (NA# negated)

The additional RESET pulse width is required to clear additional state prior to a valid self-test.

Provided the RESET falling edge meets setup and hold times t_{25} and t_{26} , the internal processor clock phase is defined at that time, as illustrated by Figure 5-28 and Figure 7-7.

A 386 DX self-test may be requested at the time RESET is negated by having the BUSY# input at a LOW level, as shown in Figure 5-28. The self-test requires $(2^{20}) +$ approximately 60 CLK2 periods to complete. The self-test duration is not affected by the test results. Even if the self-test indicates a problem, the 386 DX attempts to proceed with the reset sequence afterwards.

After the RESET falling edge (and after the self-test if it was requested) the 386 DX performs an internal initialization sequence for approximately 350 to 450 CLK2 periods.

The 386 DX samples its ERROR# input some time after the falling edge of RESET and before executing the first ESC instruction. During this sampling period BUSY# must be HIGH. If ERROR# was sampled active, the 386 DX employs the 32-bit protocol of the 387 DX. Even though this protocol was selected, it is still necessary to use a software recognition test to determine the presence or identity of the coprocessor and to assure compatibility with future processors. (See Chapter 11 of the 386™ DX Programmer's Reference Manual, Order #230985-002).

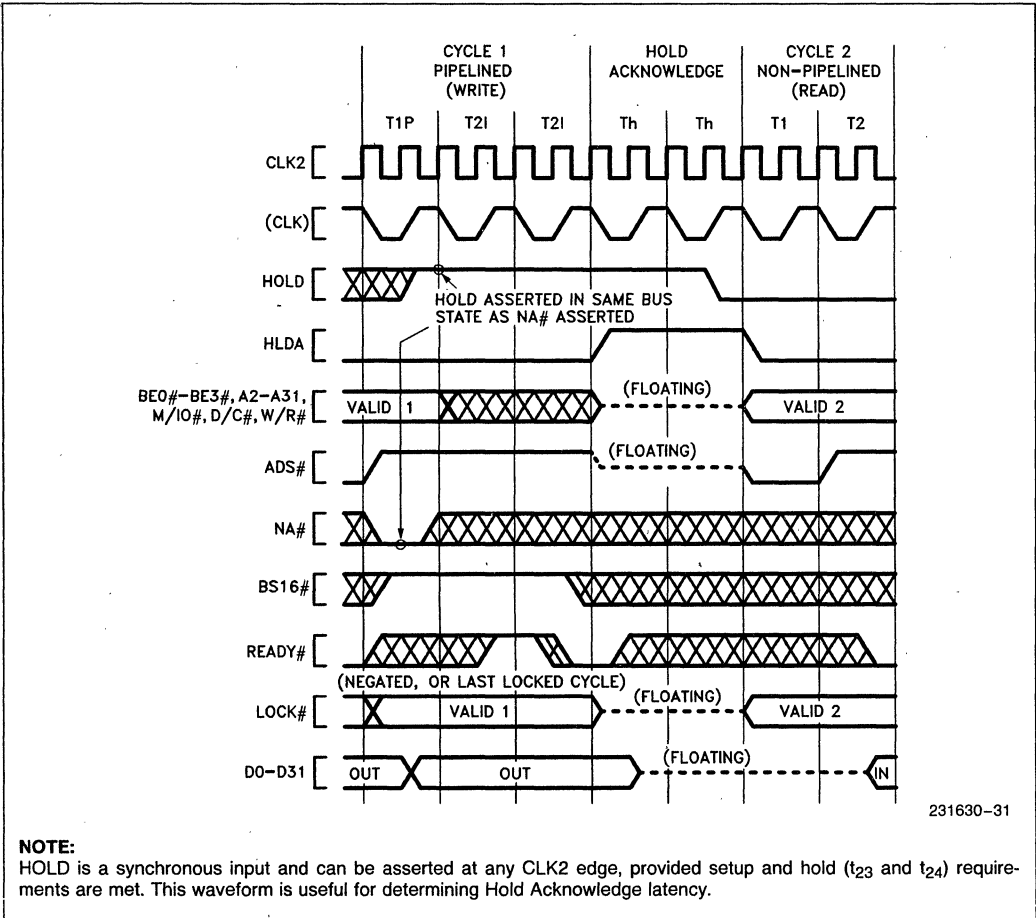


Figure 5-27. Requesting Hold from Active Bus (NA# asserted)

5.6 SELF-TEST SIGNATURE

Upon completion of self-test, (if self-test was requested by holding BUSY# LOW at least eight CLK2 periods before and after the falling edge of RESET), the EAX register will contain a signature of 00000000h indicating the 386 DX passed its self-test of microcode and major PLA contents with no problems detected. The passing signature in EAX, 00000000h, applies to all 386 DX revision levels. Any non-zero signature indicates the 386 DX unit is faulty.

5.7 COMPONENT AND REVISION IDENTIFIERS

To assist 386 DX users, the 386 DX after reset holds a component identifier and a revision identifier

in its DX register. The upper 8 bits of DX hold 03h as identification of the 386 DX component. The lower 8 bits of DX hold an 8-bit unsigned binary number related to the component revision level. The revision identifier begins chronologically with a value zero and is subject to change (typically it will be incremented) with component steppings intended to have certain improvements or distinctions from previous steppings.

These features are intended to assist 386 DX users to a practical extent. However, the revision identifier value is not guaranteed to change with every stepping revision, or to follow a completely uniform numerical sequence, depending on the type or intention of revision, or manufacturing materials required to be changed. Intel has sole discretion over these characteristics of the component.

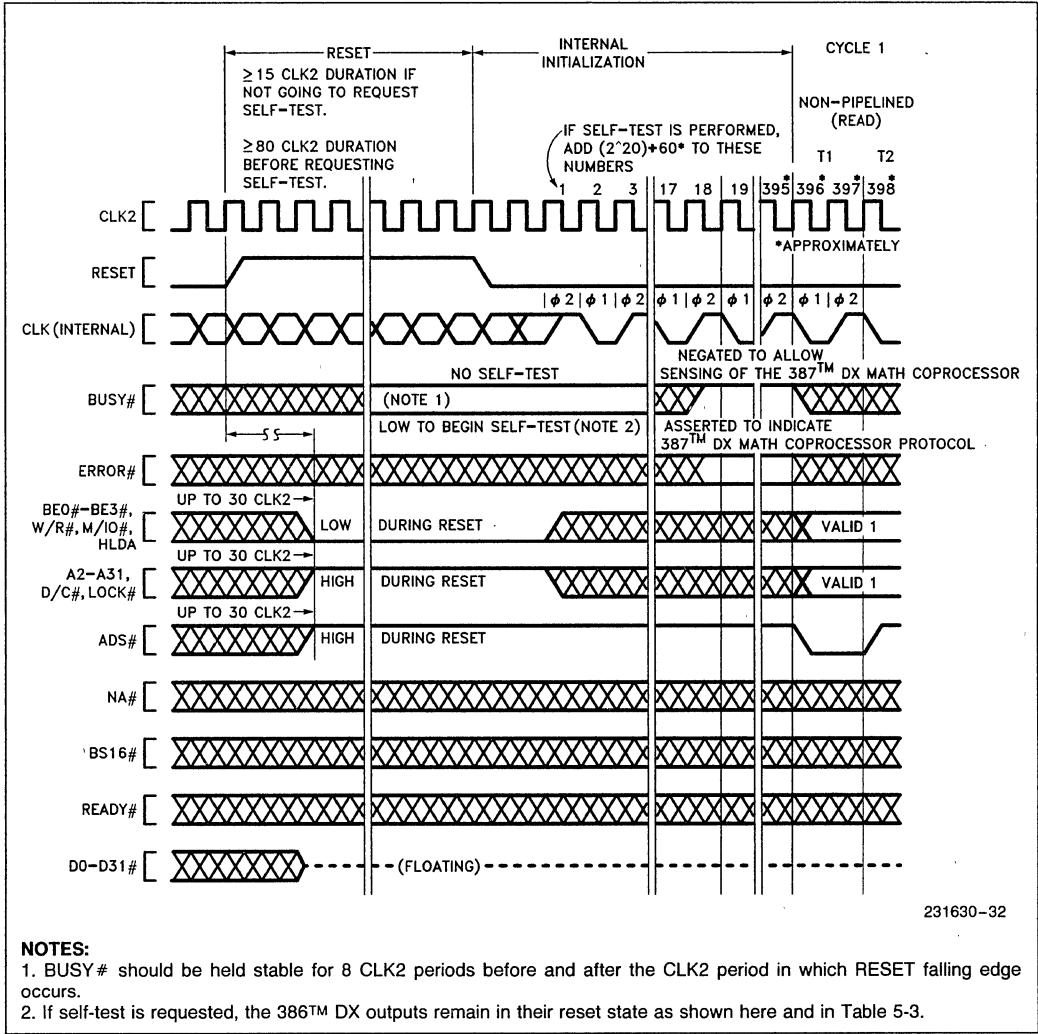


Figure 5-28. Bus Activity from Reset Until First Code Fetch

Table 5-10. Component and Revision Identifier History

386™ DX Stepping Name	Component Identifier	Revision Identifier	386™ DX Stepping Name	Component Identifier	Revision Identifier
B0	03	03	D0	03	05
B1	03	03	D1	03	08

5.8 COPROCESSOR INTERFACING

The 386 DX provides an automatic interface for the Intel 387 DX numeric floating-point coprocessor. The 387 DX coprocessor uses an I/O-mapped interface driven automatically by the 386 DX and assisted by three dedicated signals: **BUSY#**, **ERROR#**, and **PEREQ**.

As the 386 DX begins supporting a coprocessor instruction, it tests the **BUSY#** and **ERROR#** signals to determine if the coprocessor can accept its next instruction. Thus, the **BUSY#** and **ERROR#** inputs eliminate the need for any "preamble" bus cycles for communication between processor and coprocessor. The 387 DX can be given its command opcode immediately. The dedicated signals provide instruction synchronization, and eliminate the need of using the 386 DX **WAIT** opcode (9Bh) for 387 DX coprocessor instruction synchronization (the **WAIT** opcode was required when 8086 or 8088 was used with the 8087 coprocessor).

Custom coprocessors can be included in 386 DX-based systems, via memory-mapped or I/O-mapped interfaces. Such coprocessor interfaces allow a completely custom protocol, and are not limited to a set of coprocessor protocol "primitives". Instead, memory-mapped or I/O-mapped interfaces may use all applicable 386 DX instructions for high-speed coprocessor communication. The **BUSY#** and **ERROR#** inputs of the 386 DX may also be used for the custom coprocessor interface, if such hardware assist is desired. These signals can be tested by the 386 DX **WAIT** opcode (9Bh). The **WAIT** instruction will wait until the **BUSY#** input is negated (interruptable by an **NMI** or enabled **INTR** input), but generates an exception **i6 fault** if the **ERROR#** pin is in the asserted state when the **BUSY#** goes (or is) negated. If the custom coprocessor interface is memory-mapped, protection of the addresses used for the interface can be provided with the 386 DX

on-chip paging or segmentation mechanisms. If the custom interface is I/O-mapped, protection of the interface can be provided with the 386 DX **IOPL** (I/O Privilege Level) mechanism.

The 387 DX numeric coprocessor interface is I/O mapped as shown in Table 5-11. Note that the 387 DX coprocessor interface addresses are beyond the 0h-FFFFh range for programmed I/O. When the 386 DX supports the 387 DX coprocessor, the 386 DX automatically generates bus cycles to the coprocessor interface addresses.

Table 5-11. Numeric Coprocessor Port Addresses

Address in 386™ DX I/O Space	387™DX Coprocessor Register
80000F8h	Opcode Register (32-bit port)
80000FCh	Operand Register (32-bit port)

To correctly map the 387 DX coprocessor registers to the appropriate I/O addresses, connect the 387 DX coprocessor **CMD0#** pin directly to the **A2** output of the 386 DX.

5.8.1 Software Testing for Coprocessor Presence

When software is used to test for coprocessor (387 DX) presence, it should use only the following coprocessor opcodes: **FINIT**, **FNINIT**, **FSTCW mem**, **FSTSW mem**, **FSTSW AX**. To use other coprocessor opcodes when a coprocessor is known to be not present, first set **EM = 1** in 386 DX **CRO**.

6. INSTRUCTION SET

This section describes the 386 DX instruction set. A table lists all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in the following sections, which completely describe the encoding structure and the definition of all fields occurring within 386 DX instructions.

6.1 386™ DX INSTRUCTION ENCODING AND CLOCK COUNT SUMMARY

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Table 6-1 below, by the processor clock period (e.g. 50 ns for a 20 MHz 386 DX, 40 ns for a 25 MHz 386 DX, and 30 ns for a 33 MHz 386 DX).

For more detailed information on the encodings of instructions refer to section 6.2 Instruction Encodings. Section 6.2 explains the general structure of instruction encodings, and defines exactly the encodings of all fields contained within the instruction.

Instruction Clock Count Assumptions

1. The instruction has been prefetched, decoded, and is ready for execution.
2. Bus cycles do not require wait states.
3. There are no local bus HOLD requests delaying processor access to the bus.
4. No exceptions are detected during instruction execution.
5. If an effective address is calculated, it does not use two general register components. One register, scaling and displacement can be used within the clock counts shown. However, if the effective address calculation uses two general register components, add 1 clock to the clock count shown.

Instruction Clock Count Notation

1. If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand.
2. n = number of times repeated.
3. m = number of components in the next instruction executed, where the entire displacement (if any) counts as one component, the entire immediate data (if any) counts as one component, and each of the **other** bytes of the instruction and prefix(es) each count as one component.



Table 6-1. 386™ DX Instruction Set Clock Count Summary

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES					
		Real Address Mode or Virtual Address Mode	Protected Virtual Address Mode	Real Address Mode or Virtual Address Mode	Protected Virtual Address Mode				
GENERAL DATA TRANSFER									
MOV = Move:									
Register to Register/Memory	<table border="1"><tr><td>1 0 0 0 1 0 0 w</td><td>mod reg</td><td>r/m</td></tr></table>	1 0 0 0 1 0 0 w	mod reg	r/m	2/2	2/2	b	h	
1 0 0 0 1 0 0 w	mod reg	r/m							
Register/Memory to Register	<table border="1"><tr><td>1 0 0 0 1 0 1 w</td><td>mod reg</td><td>r/m</td></tr></table>	1 0 0 0 1 0 1 w	mod reg	r/m	2/4	2/4	b	h	
1 0 0 0 1 0 1 w	mod reg	r/m							
Immediate to Register/Memory	<table border="1"><tr><td>1 1 0 0 0 1 1 w</td><td>mod 0 0 0</td><td>r/m</td></tr></table> immediate data	1 1 0 0 0 1 1 w	mod 0 0 0	r/m	2/2	2/2	b	h	
1 1 0 0 0 1 1 w	mod 0 0 0	r/m							
Immediate to Register (short form)	<table border="1"><tr><td>1 0 1 1 w</td><td>reg</td></tr></table> immediate data	1 0 1 1 w	reg	2	2				
1 0 1 1 w	reg								
Memory to Accumulator (short form)	<table border="1"><tr><td>1 0 1 0 0 0 0 w</td></tr></table> full displacement	1 0 1 0 0 0 0 w	4	4	b	h			
1 0 1 0 0 0 0 w									
Accumulator to Memory (short form)	<table border="1"><tr><td>1 0 1 0 0 0 1 w</td></tr></table> full displacement	1 0 1 0 0 0 1 w	2	2	b	h			
1 0 1 0 0 0 1 w									
Register Memory to Segment Register	<table border="1"><tr><td>1 0 0 0 1 1 1 0</td><td>mod sreg3</td><td>r/m</td></tr></table>	1 0 0 0 1 1 1 0	mod sreg3	r/m	2/5	18/19	b	h, i, j	
1 0 0 0 1 1 1 0	mod sreg3	r/m							
Segment Register to Register/Memory	<table border="1"><tr><td>1 0 0 0 1 1 0 0</td><td>mod sreg3</td><td>r/m</td></tr></table>	1 0 0 0 1 1 0 0	mod sreg3	r/m	2/2	2/2	b	h	
1 0 0 0 1 1 0 0	mod sreg3	r/m							
MOVX = Move With Sign Extension									
Register From Register/Memory	<table border="1"><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 1 1 1 1 1 w</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 0 0 1 1 1 1	1 0 1 1 1 1 1 w	mod reg	r/m	3/6	3/6	b	h
0 0 0 0 1 1 1 1	1 0 1 1 1 1 1 w	mod reg	r/m						
MOVZX = Move With Zero Extension									
Register From Register/Memory	<table border="1"><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 1 1 0 1 1 w</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 0 0 1 1 1 1	1 0 1 1 0 1 1 w	mod reg	r/m	3/6	3/6	b	h
0 0 0 0 1 1 1 1	1 0 1 1 0 1 1 w	mod reg	r/m						
PUSH = Push:									
Register/Memory	<table border="1"><tr><td>1 1 1 1 1 1 1 1</td><td>mod 1 1 0</td><td>r/m</td></tr></table>	1 1 1 1 1 1 1 1	mod 1 1 0	r/m	5	5	b	h	
1 1 1 1 1 1 1 1	mod 1 1 0	r/m							
Register (short form)	<table border="1"><tr><td>0 1 0 1 0</td><td>reg</td></tr></table>	0 1 0 1 0	reg	2	2	b	h		
0 1 0 1 0	reg								
Segment Register (ES, CS, SS or DS)	<table border="1"><tr><td>0 0 0 sreg2</td><td>1 1 0</td></tr></table>	0 0 0 sreg2	1 1 0	2	2	b	h		
0 0 0 sreg2	1 1 0								
Segment Register (FS or GS)	<table border="1"><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 sreg3</td><td>0 0 0</td></tr></table>	0 0 0 0 1 1 1 1	1 0 sreg3	0 0 0	2	2	b	h	
0 0 0 0 1 1 1 1	1 0 sreg3	0 0 0							
Immediate	<table border="1"><tr><td>0 1 1 0 1 0 s 0</td></tr></table> immediate data	0 1 1 0 1 0 s 0	2	2	b	h			
0 1 1 0 1 0 s 0									
PUSHA = Push All	<table border="1"><tr><td>0 1 1 0 0 0 0 0</td></tr></table>	0 1 1 0 0 0 0 0	18	18	b	h			
0 1 1 0 0 0 0 0									
POP = Pop									
Register/Memory	<table border="1"><tr><td>1 0 0 0 1 1 1 1</td><td>mod 0 0 0</td><td>r/m</td></tr></table>	1 0 0 0 1 1 1 1	mod 0 0 0	r/m	5	5	b	h	
1 0 0 0 1 1 1 1	mod 0 0 0	r/m							
Register (short form)	<table border="1"><tr><td>0 1 0 1 1</td><td>reg</td></tr></table>	0 1 0 1 1	reg	4	4	b	h		
0 1 0 1 1	reg								
Segment Register (ES, SS or DS)	<table border="1"><tr><td>0 0 0 sreg2</td><td>1 1 1</td></tr></table>	0 0 0 sreg2	1 1 1	7	21	b	h, i, j		
0 0 0 sreg2	1 1 1								
Segment Register (FS or GS)	<table border="1"><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 sreg3</td><td>0 0 0 1</td></tr></table>	0 0 0 0 1 1 1 1	1 0 sreg3	0 0 0 1	7	21	b	h, i, j	
0 0 0 0 1 1 1 1	1 0 sreg3	0 0 0 1							
POPA = Pop All	<table border="1"><tr><td>0 1 1 0 0 0 0 1</td></tr></table>	0 1 1 0 0 0 0 1	24	24	b	h			
0 1 1 0 0 0 0 1									
XCHG = Exchange									
Register/Memory With Register	<table border="1"><tr><td>1 0 0 0 0 1 1 w</td><td>mod reg</td><td>r/m</td></tr></table>	1 0 0 0 0 1 1 w	mod reg	r/m	3/5	3/5	b, f	f, h	
1 0 0 0 0 1 1 w	mod reg	r/m							
Register With Accumulator (short form)	<table border="1"><tr><td>1 0 0 1 0</td><td>reg</td></tr></table>	1 0 0 1 0	reg	3	3				
1 0 0 1 0	reg								
IN = Input from:									
Fixed Port	<table border="1"><tr><td>1 1 1 0 0 1 0 w</td><td>port number</td></tr></table>	1 1 1 0 0 1 0 w	port number	†26	12	6*/26**	m		
1 1 1 0 0 1 0 w	port number								
Variable Port	<table border="1"><tr><td>1 1 1 0 1 1 0 w</td></tr></table>	1 1 1 0 1 1 0 w	†27	13	7*/27**	m			
1 1 1 0 1 1 0 w									
OUT = Output to:									
Fixed Port	<table border="1"><tr><td>1 1 1 0 0 1 1 w</td><td>port number</td></tr></table>	1 1 1 0 0 1 1 w	port number	†24	10	4*/24**	m		
1 1 1 0 0 1 1 w	port number								
Variable Port	<table border="1"><tr><td>1 1 1 0 1 1 1 w</td></tr></table>	1 1 1 0 1 1 1 w	†25	11	5*/25**	m			
1 1 1 0 1 1 1 w									
LEA = Load EA to Register	<table border="1"><tr><td>1 0 0 0 1 1 0 1</td><td>mod reg</td><td>r/m</td></tr></table>	1 0 0 0 1 1 0 1	mod reg	r/m	2	2			
1 0 0 0 1 1 0 1	mod reg	r/m							

* If CPL ≤ IOPL

** If CPL > IOPL

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
SEGMENT CONTROL					
LDS = Load Pointer to DS	1 1 0 0 0 1 0 1 mod reg r/m	7	22	b	h, i, j
LES = Load Pointer to ES	1 1 0 0 0 1 0 0 mod reg r/m	7	22	b	h, i, j
LFS = Load Pointer to FS	0 0 0 0 1 1 1 1 1 0 1 1 0 1 0 0 mod reg r/m	7	25	b	h, i, j
LGS = Load Pointer to GS	0 0 0 0 1 1 1 1 1 0 1 1 0 1 0 1 mod reg r/m	7	25	b	h, i, j
LSS = Load Pointer to SS	0 0 0 0 1 1 1 1 1 0 1 1 0 0 1 0 mod reg r/m	7	22	b	h, i, j
FLAG CONTROL					
CLC = Clear Carry Flag	1 1 1 1 1 0 0 0	2	2		
CLD = Clear Direction Flag	1 1 1 1 1 1 0 0	2	2		
CLI = Clear Interrupt Enable Flag	1 1 1 1 1 0 1 0	8	8		m
CLTS = Clear Task Switched Flag	0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 0	6	6	c	l
CMC = Complement Carry Flag	1 1 1 1 0 1 0 1	2	2		
LAHF = Load AH into Flag	1 0 0 1 1 1 1 1	2	2		
POPF = Pop Flags	1 0 0 1 1 1 0 1	5	5	b	h, n
PUSHF = Push Flags	1 0 0 1 1 1 0 0	4	4	b	h
SAHF = Store AH into Flags	1 0 0 1 1 1 1 0	3	3		
STC = Set Carry Flag	1 1 1 1 1 0 0 1	2	2		
STD = Set Direction Flag	1 1 1 1 1 1 0 1	2	2		
STI = Set Interrupt Enable Flag	1 1 1 1 1 0 1 1	8	8		m
ARITHMETIC					
ADD = Add					
Register to Register	0 0 0 0 0 d w mod reg r/m	2	2		
Register to Memory	0 0 0 0 0 0 w mod reg r/m	7	7	b	h
Memory to Register	0 0 0 0 0 1 w mod reg r/m	6	6	b	h
Immediate to Register/Memory	1 0 0 0 0 s w mod 0 0 0 r/m immediate data	2/7	2/7	b	h
Immediate to Accumulator (short form)	0 0 0 0 1 0 w immediate data	2	2		
ADC = Add With Carry					
Register to Register	0 0 0 1 0 0 d w mod reg r/m	2	2		
Register to Memory	0 0 0 1 0 0 0 w mod reg r/m	7	7	b	h
Memory to Register	0 0 0 1 0 0 1 w mod reg r/m	6	6	b	h
Immediate to Register/Memory	1 0 0 0 0 s w mod 0 1 0 r/m immediate data	2/7	2/7	b	h
Immediate to Accumulator (short form)	0 0 0 1 0 1 0 w immediate data	2	2		
INC = Increment					
Register/Memory	1 1 1 1 1 1 1 w mod 0 0 0 r/m	2/6	2/6	b	h
Register (short form)	0 1 0 0 0 reg	2	2		
SUB = Subtract					
Register from Register	0 0 1 0 1 0 d w mod reg r/m	2	2		

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
ARITHMETIC (Continued)					
Register from Memory	0010100w mod reg r/m	7	7	b	h
Memory from Register	0010101w mod reg r/m	6	6	b	h
Immediate from Register/Memory	100000sw mod 101 r/m immediate data	2/7	2/7	b	h
Immediate from Accumulator (short form)	0010110w immediate data	2	2		
SBB = Subtract with Borrow					
Register from Register	000110dw mod reg r/m	2	2		
Register from Memory	0001100w mod reg r/m	7	7	b	h
Memory from Register	0001101w mod reg r/m	6	6	b	h
Immediate from Register/Memory	100000sw mod 011 r/m immediate data	2/7	2/7	b	h
Immediate from Accumulator (short form)	0001110w immediate data	2	2		
DEC = Decrement					
Register/Memory	1111111w reg 001 r/m	2/6	2/6	b	h
Register (short form)	01001 reg	2	2		
CMP = Compare					
Register with Register	001110dw mod reg r/m	2	2		
Memory with Register	0011100w mod reg r/m	5	5	b	h
Register with Memory	0011101w mod reg r/m	6	6	b	h
Immediate with Register/Memory	100000sw mod 111 r/m immediate data	2/5	2/5	b	h
Immediate with Accumulator (short form)	0011110w immediate data	2	2		
NEG = Change Sign					
	1111011w mod 011 r/m	2/6	2/6	b	h
AAA = ASCII Adjust for Add					
	00110111	4	4		
AAS = ASCII Adjust for Subtract					
	00111111	4	4		
DAA = Decimal Adjust for Add					
	00100111	4	4		
DAS = Decimal Adjust for Subtract					
	00101111	4	4		
MUL = Multiply (unsigned)					
Accumulator with Register/Memory	1111011w mod 100 r/m				
Multiplier-Byte		12-17/15-20	12-17/15-20	b, d	d, h
-Word		12-25/15-28	12-25/15-28	b, d	d, h
-Doubleword		12-41/15-44	12-41/15-44	b, d	d, h
IMUL = Integer Multiply (signed)					
Accumulator with Register/Memory	1111011w mod 101 r/m				
Multiplier-Byte		12-17/15-20	12-17/15-20	b, d	d, h
-Word		12-25/15-28	12-25/15-28	b, d	d, h
-Doubleword		12-41/15-44	12-41/15-44	b, d	d, h
Register with Register/Memory	00001111 10101111 mod reg r/m				
Multiplier-Byte		12-17/15-20	12-17/15-20	b, d	d, h
-Word		12-25/15-28	12-25/15-28	b, d	d, h
-Doubleword		12-41/15-44	12-41/15-44	b, d	d, h
Register/Memory with Immediate to Register	011010s1 mod reg r/m immediate data				
-Word		13-26/14-27	13-26/14-27	b, d	d, h
-Doubleword		13-42/14-43	13-42/14-43	b, d	d, h

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
ARITHMETIC (Continued)					
DIV = Divide (Unsigned)					
Accumulator by Register/Memory	1111011w mod 110 r/m				
Divisor—Byte		14/17	14/17	b,e	e,h
—Word		22/25	22/25	b,e	e,h
—Doubleword		38/41	38/41	b,e	e,h
IDIV = Integer Divide (Signed)					
Accumulator By Register/Memory	1111011w mod 111 r/m				
Divisor—Byte		19/22	19/22	b,e	e,h
—Word		27/30	27/30	b,e	e,h
—Doubleword		43/46	43/46	b,e	e,h
AAD = ASCII Adjust for Divide	11010101 00001010	19	19		
AAM = ASCII Adjust for Multiply	11010100 00001010	17	17		
CBW = Convert Byte to Word	10011000	3	3		
CWD = Convert Word to Double Word	10011001	2	2		
LOGIC					
Shift Rotate Instructions					
Not Through Carry (ROL, ROR, SAL, SAR, SHL, and SHR)					
Register/Memory by 1	1101000w mod TTT r/m	3/7	3/7	b	h
Register/Memory by CL	1101001w mod TTT r/m	3/7	3/7	b	h
Register/Memory by Immediate Count	1100000w mod TTT r/m	3/7	3/7	b	h
Through Carry (RCL and RCR)					
Register/Memory by 1	1101000w mod TTT r/m	9/10	9/10	b	h
Register/Memory by CL	1101001w mod TTT r/m	9/10	9/10	b	h
Register/Memory by Immediate Count	1100000w mod TTT r/m	9/10	9/10	b	h
	TTT Instruction 000 ROL 001 ROR 010 RCL 011 RCR 100 SHL/SAL 101 SHR 111 SAR				
SHLD = Shift Left Double					
Register/Memory by Immediate	00001111 10100100 mod reg r/m	3/7	3/7		immed 8-bit data
Register/Memory by CL	00001111 10100101 mod reg r/m	3/7	3/7		
SHRD = Shift Right Double					
Register/Memory by Immediate	00001111 10101100 mod reg r/m	3/7	3/7		immed 8-bit data
Register/Memory by CL	00001111 10101101 mod reg r/m	3/7	3/7		
AND = And					
Register to Register	001000dw mod reg r/m	2	2		



386™ DX MICROPROCESSOR

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
LOGIC (Continued)					
Register to Memory	0010000w mod reg r/m	7	7	b	h
Memory to Register	0010001w mod reg r/m	6	6	b	h
Immediate to Register/Memory	1000000w mod 100 r/m immediate data	2/7	2/7	b	h
Immediate to Accumulator (Short Form)	0010010w immediate data	2	2		
TEST = And Function to Flags, No Result					
Register/Memory and Register	1000010w mod reg r/m	2/5	2/5	b	h
Immediate Data and Register/Memory	1111011w mod 000 r/m immediate data	2/5	2/5	b	h
Immediate Data and Accumulator (Short Form)	1010100w immediate data	2	2		
OR = Or					
Register to Register	000010dw mod reg r/m	2	2		
Register to Memory	0000100w mod reg r/m	7	7	b	h
Memory to Register	0000101w mod reg r/m	6	6	b	h
Immediate to Register/Memory	1000000w mod 001 r/m immediate data	2/7	2/7	b	h
Immediate to Accumulator (Short Form)	0000110w immediate data	2	2		
XOR = Exclusive Or					
Register to Register	001100dw mod reg r/m	2	2		
Register to Memory	0011000w mod reg r/m	7	7	b	h
Memory to Register	0011001w mod reg r/m	6	6	b	h
Immediate to Register/Memory	1000000w mod 110 r/m immediate data	2/7	2/7	b	h
Immediate to Accumulator (Short Form)	0011010w immediate data	2	2		
NOT = Invert Register/Memory	1111011w mod 010 r/m	2/6	2/6	b	h
STRING MANIPULATION					
CMPS = Compare Byte Word	1010011w	10	10	b	h
INS = Input Byte/Word from DX Port	0110110w	†29	9*/29**	b	h,m
LODS = Load Byte/Word to AL/AX/EAX	1010110w	5	5	b	h
MOVS = Move Byte Word	1010010w	8	8	b	h
OUTS = Output Byte/Word to DX Port	0110111w	†28	8*/28**	b	h,m
SCAS = Scan Byte Word	1010111w	8	8	b	h
STOS = Store Byte/Word from AL/AX/EX	1010101w	5	5	b	h
XLAT = Translate String	11010111	5	5		h
REPEATED STRING MANIPULATION					
Repeated by Count in CX or ECX					
REPE CMPS = Compare String	11110011 1010011w	5+9n	5+9n	b	h
(Find Non-Match)					

* If CPL ≤ IOPL

** If CPL > IOPL

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT	NOTES								
			Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode					
REPEATED STRING MANIPULATION (Continued)											
REPNE CMPS = Compare String (Find Match)	<table border="1"><tr><td>11110010</td><td>1010011w</td></tr></table>	11110010	1010011w	Clk Count Virtual 8086 Mode	5+9n	5+9n	b	h			
11110010	1010011w										
REP INS = Input String	<table border="1"><tr><td>11110010</td><td>0110110w</td></tr></table>	11110010	0110110w	†28+6n	14+6n	8+6n*/28+6n**	b	h, m			
11110010	0110110w										
REP LODS = Load String	<table border="1"><tr><td>11110010</td><td>1010110w</td></tr></table>	11110010	1010110w		5+6n	5+6n	b	h			
11110010	1010110w										
REP MOVS = Move String	<table border="1"><tr><td>11110010</td><td>1010010w</td></tr></table>	11110010	1010010w		8+4n	8+4n	b	h			
11110010	1010010w										
REP OUTS = Output String	<table border="1"><tr><td>11110010</td><td>0110111w</td></tr></table>	11110010	0110111w	†26+5n	12+5n	6+5n*/26+5n**	b	h, m			
11110010	0110111w										
REPE SCAS = Scan String (Find Non-AL/AX/EAX)	<table border="1"><tr><td>11110011</td><td>1010111w</td></tr></table>	11110011	1010111w		5+8n	5+8n	b	h			
11110011	1010111w										
REPNE SCAS = Scan String (Find AL/AX/EAX)	<table border="1"><tr><td>11110010</td><td>1010111w</td></tr></table>	11110010	1010111w		5+8n	5+8n	b	h			
11110010	1010111w										
REP STOS = Store String	<table border="1"><tr><td>11110010</td><td>1010101w</td></tr></table>	11110010	1010101w		5+5n	5+5n	b	h			
11110010	1010101w										
BIT MANIPULATION											
BSF = Scan Bit Forward	<table border="1"><tr><td>00001111</td><td>10111100</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	10111100	mod reg	r/m		11+3n	11+3n	b	h	
00001111	10111100	mod reg	r/m								
BSR = Scan Bit Reverse	<table border="1"><tr><td>00001111</td><td>10111101</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	10111101	mod reg	r/m		9+3n	9+3n	b	h	
00001111	10111101	mod reg	r/m								
BT = Test Bit											
Register/Memory, Immediate	<table border="1"><tr><td>00001111</td><td>10111010</td><td>mod 100</td><td>r/m</td><td>immed 8-bit data</td></tr></table>	00001111	10111010	mod 100	r/m	immed 8-bit data		3/6	3/6	b	h
00001111	10111010	mod 100	r/m	immed 8-bit data							
Register/Memory, Register	<table border="1"><tr><td>00001111</td><td>10100011</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	10100011	mod reg	r/m		3/12	3/12	b	h	
00001111	10100011	mod reg	r/m								
BTC = Test Bit and Complement											
Register/Memory, Immediate	<table border="1"><tr><td>00001111</td><td>10111010</td><td>mod 111</td><td>r/m</td><td>immed 8-bit data</td></tr></table>	00001111	10111010	mod 111	r/m	immed 8-bit data		6/8	6/8	b	h
00001111	10111010	mod 111	r/m	immed 8-bit data							
Register/Memory, Register	<table border="1"><tr><td>00001111</td><td>10111011</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	10111011	mod reg	r/m		6/13	6/13	b	h	
00001111	10111011	mod reg	r/m								
BTR = Test Bit and Reset											
Register/Memory, Immediate	<table border="1"><tr><td>00001111</td><td>10111010</td><td>mod 110</td><td>r/m</td><td>immed 8-bit data</td></tr></table>	00001111	10111010	mod 110	r/m	immed 8-bit data		6/8	6/8	b	h
00001111	10111010	mod 110	r/m	immed 8-bit data							
Register/Memory, Register	<table border="1"><tr><td>00001111</td><td>10110011</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	10110011	mod reg	r/m		6/13	6/13	b	h	
00001111	10110011	mod reg	r/m								
BTS = Test Bit and Set											
Register/Memory, Immediate	<table border="1"><tr><td>00001111</td><td>10111010</td><td>mod 101</td><td>r/m</td><td>immed 8-bit data</td></tr></table>	00001111	10111010	mod 101	r/m	immed 8-bit data		6/8	6/8	b	h
00001111	10111010	mod 101	r/m	immed 8-bit data							
Register/Memory, Register	<table border="1"><tr><td>00001111</td><td>10101011</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	10101011	mod reg	r/m		6/13	6/13	b	h	
00001111	10101011	mod reg	r/m								
CONTROL TRANSFER											
CALL = Call											
Direct Within Segment	<table border="1"><tr><td>11101000</td><td>full displacement</td></tr></table>	11101000	full displacement		7+m	7+m	b	r			
11101000	full displacement										
Register/Memory											
Indirect Within Segment	<table border="1"><tr><td>11111111</td><td>mod 010</td><td>r/m</td></tr></table>	11111111	mod 010	r/m		7+m/ 10+m	7+m/ 10+m	b	h, r		
11111111	mod 010	r/m									
Direct Intersegment	<table border="1"><tr><td>10011010</td><td>unsigned full offset, selector</td></tr></table>	10011010	unsigned full offset, selector		17+m	34+m	b	j, k, r			
10011010	unsigned full offset, selector										

NOTES:

† Clock count shown applies if I/O permission allows I/O to the port in virtual 8086 mode. If I/O bit map denies permission exception 13 fault occurs; refer to clock counts for INT 3 instruction.

* If CPL ≤ IOPL

** If CPL > IOPL



Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES				
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode			
CONTROL TRANSFER (Continued)								
Protected Mode Only (Direct Intersegment)								
	Via Call Gate to Same Privilege Level		52 + m		h,j,k,r			
	Via Call Gate to Different Privilege Level, (No Parameters)		86 + m		h,j,k,r			
	Via Call Gate to Different Privilege Level, (x Parameters)		94 + 4x + m		h,j,k,r			
	From 80286 Task to 80286 TSS		273		h,j,k,r			
	From 80286 Task to 386™ DX TSS		298		h,j,k,r			
	From 80286 Task to Virtual 8086 Task (386™ DX TSS)		218		h,j,k,r			
	From 386™ DX Task to 80286 TSS		273		h,j,k,r			
	From 386™ DX Task to 386™ DX TSS		300		h,j,k,r			
	From 386™ DX Task to Virtual 8086 Task (386™ DX TSS)		218		h,j,k,r			
Indirect Intersegment	<table border="1"><tr><td>11111111</td><td>mod 011</td><td>r/m</td></tr></table>	11111111	mod 011	r/m	22 + m	38 + m	b	h,j,k,r
11111111	mod 011	r/m						
Protected Mode Only (Indirect Intersegment)								
	Via Call Gate to Same Privilege Level		56 + m		h,j,k,r			
	Via Call Gate to Different Privilege Level, (No Parameters)		90 + m		h,j,k,r			
	Via Call Gate to Different Privilege Level, (x Parameters)		98 + 4x + m		h,j,k,r			
	From 80286 Task to 80286 TSS		278		h,j,k,r			
	From 80286 Task to 386™ DX TSS		303		h,j,k,r			
	From 80286 Task to Virtual 8086 Task (386™ DX TSS)		222		h,j,k,r			
	From 386™ DX Task to 80286 TSS		278		h,j,k,r			
	From 386™ DX Task to 386™ DX TSS		305		h,j,k,r			
	From 386™ DX Task to Virtual 8086 Task (386™ DX TSS)		222		h,j,k,r			
JMP = Unconditional Jump								
Short	<table border="1"><tr><td>11101011</td><td>8-bit displacement</td></tr></table>	11101011	8-bit displacement	7 + m	7 + m		r	
11101011	8-bit displacement							
Direct within Segment	<table border="1"><tr><td>11101001</td><td>full displacement</td></tr></table>	11101001	full displacement	7 + m	7 + m		r	
11101001	full displacement							
Register/Memory Indirect within Segment	<table border="1"><tr><td>11111111</td><td>mod 100</td><td>r/m</td></tr></table>	11111111	mod 100	r/m	7 + m/ 10 + m	7 + m/ 10 + m	b	h,r
11111111	mod 100	r/m						
Direct Intersegment	<table border="1"><tr><td>11101010</td><td>unsigned full offset, selector</td></tr></table>	11101010	unsigned full offset, selector	12 + m	27 + m		j,k,r	
11101010	unsigned full offset, selector							
Protected Mode Only (Direct Intersegment)								
	Via Call Gate to Same Privilege Level		45 + m		h,j,k,r			
	From 80286 Task to 80286 TSS		274		h,j,k,r			
	From 80286 Task to 386™ DX TSS		301		h,j,k,r			
	From 80286 Task to Virtual 8086 Task (386™ DX TSS)		219		h,j,k,r			
	From 386™ DX Task to 80286 TSS		270		h,j,k,r			
	From 386™ DX Task to 386™ DX TSS		303		h,j,k,r			
	From 386™ DX Task to Virtual 8086 Task (386™ DX TSS)		221		h,j,k,r			
Indirect Intersegment	<table border="1"><tr><td>11111111</td><td>mod 101</td><td>r/m</td></tr></table>	11111111	mod 101	r/m	17 + m	31 + m	b	h,j,k,r
11111111	mod 101	r/m						
Protected Mode Only (Indirect Intersegment)								
	Via Call Gate to Same Privilege Level		49 + m		h,j,k,r			
	From 80286 Task to 80286 TSS		279		h,j,k,r			
	From 80286 Task to 386™ DX TSS		306		h,j,k,r			
	From 80286 Task to Virtual 8086 Task (386™ DX TSS)		223		h,j,k,r			
	From 386™ DX Task to 80286 TSS		275		h,j,k,r			
	From 386™ DX Task to 386™ DX TSS		308		h,j,k,r			
	From 386™ DX Task to Virtual 8086 Task (386™ DX TSS)		225		h,j,k,r			



Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
CONTROL TRANSFER (Continued)					
RET = Return from CALL:					
Within Segment	11000011	10 + m	10 + m	b	g, h, r
Within Segment Adding Immediate to SP	11000010 16-bit displ	10 + m	10 + m	b	g, h, r
Intersegment	11001011	18 + m	32 + m	b	g, h, j, k, r
Intersegment Adding Immediate to SP	11001010 16-bit displ	18 + m	32 + m	b	g, h, j, k, r
Protected Mode Only (RET): to Different Privilege Level Intersegment Intersegment Adding Immediate to SP			69 69		h, j, k, r h, j, k, r
CONDITIONAL JUMPS					
NOTE: Times Are Jump "Taken or Not Taken"					
JO = Jump on Overflow					
8-Bit Displacement	01110000 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 10000000 full displacement	7 + m or 3	7 + m or 3		r
JNO = Jump on Not Overflow					
8-Bit Displacement	01110001 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 10000001 full displacement	7 + m or 3	7 + m or 3		r
JB/JNAE = Jump on Below/Not Above or Equal					
8-Bit Displacement	01110010 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 10000010 full displacement	7 + m or 3	7 + m or 3		r
JNB/JAE = Jump on Not Below/Above or Equal					
8-Bit Displacement	01110011 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 10000011 full displacement	7 + m or 3	7 + m or 3		r
JE/JZ = Jump on Equal/Zero					
8-Bit Displacement	01110100 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 10000100 full displacement	7 + m or 3	7 + m or 3		r
JNE/JNZ = Jump on Not Equal/Not Zero					
8-Bit Displacement	01110101 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 10000101 full displacement	7 + m or 3	7 + m or 3		r
JBE/JNA = Jump on Below or Equal/Not Above					
8-Bit Displacement	01110110 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 10000110 full displacement	7 + m or 3	7 + m or 3		r
JNBE/JA = Jump on Not Below or Equal/Above					
8-Bit Displacement	01110111 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 10000111 full displacement	7 + m or 3	7 + m or 3		r
JS = Jump on Sign					
8-Bit Displacement	01111000 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 10001000 full displacement	7 + m or 3	7 + m or 3		r

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
CONDITIONAL JUMPS (Continued)					
JNS = Jump on Not Sign					
8-Bit Displacement	0 1 1 1 1 0 0 1 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 1 0 0 1 full displacement	7 + m or 3	7 + m or 3		r
JP/JPE = Jump on Parity/Parity Even					
8-Bit Displacement	0 1 1 1 1 0 1 0 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 1 0 1 0 full displacement	7 + m or 3	7 + m or 3		r
JNP/JPO = Jump on Not Parity/Parity Odd					
8-Bit Displacement	0 1 1 1 1 0 1 1 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 1 0 1 1 full displacement	7 + m or 3	7 + m or 3		r
JL/JNGE = Jump on Less/Not Greater or Equal					
8-Bit Displacement	0 1 1 1 1 1 0 0 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 1 1 0 0 full displacement	7 + m or 3	7 + m or 3		r
JNL/JGE = Jump on Not Less/Greater or Equal					
8-Bit Displacement	0 1 1 1 1 1 0 1 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 1 1 0 1 full displacement	7 + m or 3	7 + m or 3		r
JLE/JNG = Jump on Less or Equal/Not Greater					
8-Bit Displacement	0 1 1 1 1 1 1 0 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 1 1 1 0 full displacement	7 + m or 3	7 + m or 3		r
JNLE/JG = Jump on Not Less or Equal/Greater					
8-Bit Displacement	0 1 1 1 1 1 1 1 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 full displacement	7 + m or 3	7 + m or 3		r
JCXZ = Jump on CX Zero					
	1 1 1 0 0 0 1 1 8-bit displ	9 + m or 5	9 + m or 5		r
JECXZ = Jump on ECX Zero					
	1 1 1 0 0 0 1 1 8-bit displ	9 + m or 5	9 + m or 5		r
(Address Size Prefix Differentiates JCXZ from JECXZ)					
LOOP = Loop CX Times					
	1 1 1 0 0 0 1 0 8-bit displ	11 + m	11 + m		r
LOOPZ/LOOPE = Loop with Zero/Equal					
	1 1 1 0 0 0 0 1 8-bit displ	11 + m	11 + m		r
LOOPNZ/LOOPNE = Loop While Not Zero					
	1 1 1 0 0 0 0 0 8-bit displ	11 + m	11 + m		r
CONDITIONAL BYTE SET					
NOTE: Times Are Register/Memory					
SETO = Set Byte on Overflow					
To Register/Memory	0 0 0 0 1 1 1 1 1 0 0 1 0 0 0 0 mod 0 0 0 r/m	4/5	4/5		h
SETNO = Set Byte on Not Overflow					
To Register/Memory	0 0 0 0 1 1 1 1 1 0 0 1 0 0 0 1 mod 0 0 0 r/m	4/5	4/5		h
SETB/SETNAE = Set Byte on Below/Not Above or Equal					
To Register/Memory	0 0 0 0 1 1 1 1 1 0 0 1 0 0 1 0 mod 0 0 0 r/m	4/5	4/5		h



Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
CONDITIONAL BYTE SET (Continued)					
SETNB = Set Byte on Not Below/Above or Equal					
To Register/Memory	00001111 10010011 mod 000 r/m	4/5	4/5		h
SETE/SETZ = Set Byte on Equal/Zero					
To Register/Memory	00001111 10010100 mod 000 r/m	4/5	4/5		h
SETNE/SETNZ = Set Byte on Not Equal/Not Zero					
To Register/Memory	00001111 10010101 mod 000 r/m	4/5	4/5		h
SETBE/SETNA = Set Byte on Below or Equal/Not Above					
To Register/Memory	00001111 10010110 mod 000 r/m	4/5	4/5		h
SETNBE/SETA = Set Byte on Not Below or Equal/Above					
To Register/Memory	00001111 10010111 mod 000 r/m	4/5	4/5		h
SETS = Set Byte on Sign					
To Register/Memory	00001111 10011000 mod 000 r/m	4/5	4/5		h
SETNS = Set Byte on Not Sign					
To Register/Memory	00001111 10011001 mod 000 r/m	4/5	4/5		h
SETP/SETPE = Set Byte on Parity/Parity Even					
To Register/Memory	00001111 10011010 mod 000 r/m	4/5	4/5		h
SETNP/SETPO = Set Byte on Not Parity/Parity Odd					
To Register/Memory	00001111 10011011 mod 000 r/m	4/5	4/5		h
SETL/SETNGE = Set Byte on Less/Not Greater or Equal					
To Register/Memory	00001111 10011100 mod 000 r/m	4/5	4/5		h
SETNL/SETGE = Set Byte on Not Less/Greater or Equal					
To Register/Memory	00001111 01111101 mod 000 r/m	4/5	4/5		h
SETLE/SETNG = Set Byte on Less or Equal/Not Greater					
To Register/Memory	00001111 10011110 mod 000 r/m	4/5	4/5		h
SETNLE/SETG = Set Byte on Not Less or Equal/Greater					
To Register/Memory	00001111 10011111 mod 000 r/m	4/5	4/5		h
ENTER = Enter Procedure	11001000 16-bit displacement, 8-bit level				
L = 0		10	10	b	h
L = 1		12	12	b	h
L > 1		15 +	15 +	b	h
		4(n - 1)	4(n - 1)		
LEAVE = Leave Procedure	11001001	4	4	b	h



Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
INTERRUPT INSTRUCTIONS					
INT = Interrupt:					
Type Specified	11001101 type	37		b	
Type 3	11001100	33		b	
INTO = Interrupt 4 if Overflow Flag Set	11001110				
If OF = 1		35		b, e	
If OF = 0		3	3	b, e	
Bound = Interrupt 5 if Detect Value Out of Range	01100010 mod reg r/m				
If Out of Range		44		b, e	e, g, h, j, k, r
If In Range		10	10	b, e	e, g, h, j, k, r
Protected Mode Only (INT)					
INT: Type Specified					
Via Interrupt or Trap Gate to Same Privilege Level			59		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			99		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate		282			g, j, k, r
From 80286 Task to 386™ DX TSS via Task Gate		309			g, j, k, r
From 80286 Task to virt 8086 md via Task Gate		226			g, j, k, r
From 386™ DX Task to 80286 TSS via Task Gate		284			g, j, k, r
From 386™ DX Task to 386™ DX TSS via Task Gate		311			g, j, k, r
From 386™ DX Task to virt 8086 md via Task Gate		228			g, j, k, r
From virt 8086 md to 80286 TSS via Task Gate		289			g, j, k, r
From virt 8086 md to 386™ DX TSS via Task Gate		316			g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate		119			
INT: TYPE 3					
Via Interrupt or Trap Gate to Same Privilege Level			59		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			99		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate		278			g, j, k, r
From 80286 Task to 386™ DX TSS via Task Gate		305			g, j, k, r
From 80286 Task to Virt 8086 md via Task Gate		222			g, j, k, r
From 386™ DX Task to 80286 TSS via Task Gate		280			g, j, k, r
From 386™ DX Task to 386™ DX TSS via Task Gate		307			g, j, k, r
From 386™ DX Task to Virt 8086 md via Task Gate		224			g, j, k, r
From virt 8086 md to 80286 TSS via Task Gate		285			g, j, k, r
From virt 8086 md to 386™ DX TSS via Task Gate		312			g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate		119			
INTO:					
Via Interrupt or Trap Gate to Same Privilege Level			59		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			99		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate		280			g, j, k, r
From 80286 Task to 386™ DX TSS via Task Gate		307			g, j, k, r
From 80286 Task to virt 8086 md via Task Gate		224			g, j, k, r
From 386™ DX Task to 80286 TSS via Task Gate		282			g, j, k, r
From 386™ DX Task to 386™ DX TSS via Task Gate		309			g, j, k, r
From 386™ DX Task to 386™ DX TSS via Task Gate		225			g, j, k, r
From virt 8086 md to 80286 TSS via Task Gate		287			g, j, k, r
From virt 8086 md to 386™ DX TSS via Task Gate		314			g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate		119			

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES			
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode		
INTERRUPT INSTRUCTIONS (Continued)							
BOUND:							
Via Interrupt or Trap Gate to Same Privilege Level			59		g, j, k, r		
Via Interrupt or Trap Gate to Different Privilege Level			99		g, j, k, r		
From 80286 Task to 80286 TSS via Task Gate			254		g, j, k, r		
From 80286 Task to 386™ DX TSS via Task Gate			284		g, j, k, r		
From 80286 Task to virt 8086 Mode via Task Gate			231		g, j, k, r		
From 386™ DX Task to 80286 TSS via Task Gate			264		g, j, k, r		
From 386™ DX Task to 386™ DX TSS via Task Gate			294		g, j, k, r		
From 80368 Task to virt 8086 Mode via Task Gate			243		g, j, k, r		
From virt 8086 Mode to 80286 TSS via Task Gate			264		g, j, k, r		
From virt 8086 Mode to 386™ DX TSS via Task Gate			294		g, j, k, r		
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate			119				
INTERRUPT RETURN							
IRET = Interrupt Return	<table border="1"><tr><td>11001111</td></tr></table>	11001111		22		g, h, j, k, r	
11001111							
Protected Mode Only (IRET)							
To the Same Privilege Level (within task)			38		g, h, j, k, r		
To Different Privilege Level (within task)			82		g, h, j, k, r		
From 80286 Task to 80286 TSS			232		h, j, k, r		
From 80286 Task to 386™ DX TSS			265		h, j, k, r		
From 80286 Task to Virtual 8086 Task			213		h, j, k, r		
From 80286 Task to Virtual 8086 Mode (within task)			60				
From 386™ DX Task to 80286 TSS			271		h, j, k, r		
From 386™ DX Task to 386™ DX TSS			275		h, j, k, r		
From 386™ DX Task to Virtual 8086 Task			223		h, j, k, r		
From 386™ DX Task to Virtual 8086 Mode (within task)			60				
PROCESSOR CONTROL							
HLT = HALT	<table border="1"><tr><td>11110100</td></tr></table>	11110100		5	5	l	
11110100							
MOV = Move to and From Control/Debug/Test Registers							
CR0/CR2/CR3 from register	<table border="1"><tr><td>00001111</td><td>00100010</td><td>11eee reg</td></tr></table>	00001111	00100010	11eee reg	11/4/5	11/4/5	l
00001111	00100010	11eee reg					
Register From CR0-3	<table border="1"><tr><td>00001111</td><td>00100000</td><td>11eee reg</td></tr></table>	00001111	00100000	11eee reg	6	6	l
00001111	00100000	11eee reg					
DR0-3 From Register	<table border="1"><tr><td>00001111</td><td>00100011</td><td>11eee reg</td></tr></table>	00001111	00100011	11eee reg	22	22	l
00001111	00100011	11eee reg					
DR6-7 From Register	<table border="1"><tr><td>00001111</td><td>00100011</td><td>11eee reg</td></tr></table>	00001111	00100011	11eee reg	16	16	l
00001111	00100011	11eee reg					
Register from DR6-7	<table border="1"><tr><td>00001111</td><td>00100001</td><td>11eee reg</td></tr></table>	00001111	00100001	11eee reg	14	14	l
00001111	00100001	11eee reg					
Register from DR0-3	<table border="1"><tr><td>00001111</td><td>00100001</td><td>11eee reg</td></tr></table>	00001111	00100001	11eee reg	22	22	l
00001111	00100001	11eee reg					
TR6-7 from Register	<table border="1"><tr><td>00001111</td><td>00100110</td><td>11eee reg</td></tr></table>	00001111	00100110	11eee reg	12	12	l
00001111	00100110	11eee reg					
Register from TR6-7	<table border="1"><tr><td>00001111</td><td>00100100</td><td>11eee reg</td></tr></table>	00001111	00100100	11eee reg	12	12	l
00001111	00100100	11eee reg					
NOP = No Operation	<table border="1"><tr><td>10010000</td></tr></table>	10010000		3	3		
10010000							
WAIT = Wait until BUSY# pin is negated	<table border="1"><tr><td>10011011</td></tr></table>	10011011		7	7		
10011011							



Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
PROCESSOR EXTENSION INSTRUCTIONS					
Processor Extension Escape	11011TTT mod LLL r/m TTT and LLL bits are opcode information for coprocessor.			See 80287/80387 data sheets for clock counts	h
PREFIX BYTES					
Address Size Prefix	01100111	0	0		
LOCK = Bus Lock Prefix	11110000	0	0		m
Operand Size Prefix	01100110	0	0		
Segment Override Prefix					
CS:	00101110	0	0		
DS:	00111110	0	0		
ES:	00100110	0	0		
FS:	01100100	0	0		
GS:	01100101	0	0		
SS:	00110110	0	0		
PROTECTION CONTROL					
ARPL = Adjust Requested Privilege Level					
From Register/Memory	01100011 mod reg r/m	N/A	20/21	a	h
LAR = Load Access Rights					
From Register/Memory	00001111 00000010 mod reg r/m	N/A	15/16	a	g, h, j, p
LGDT = Load Global Descriptor					
Table Register	00001111 00000001 mod 010 r/m	11	11	b, c	h, i
LIDT = Load Interrupt Descriptor					
Table Register	00001111 00000001 mod 011 r/m	11	11	b, c	i, i
LLDT = Load Local Descriptor					
Table Register to Register/Memory	00001111 00000000 mod 010 r/m	N/A	20/24	a	g, h, j, l
LMSW = Load Machine Status Word					
From Register/Memory	00001111 00000001 mod 110 r/m	11/14	11/14	b, c	h, l
LSL = Load Segment Limit					
From Register/Memory	00001111 00000011 mod reg r/m				
Byte-Granular Limit		N/A	21/22	a	g, h, j, p
Page-Granular Limit		N/A	25/26	a	g, h, j, p
LTR = Load Task Register					
From Register/Memory	00001111 00000000 mod 001 r/m	N/A	23/27	a	g, h, j, l
SGDT = Store Global Descriptor					
Table Register	00001111 00000001 mod 000 r/m	9	9	b, c	h
SIDT = Store Interrupt Descriptor					
Table Register	00001111 00000001 mod 001 r/m	9	9	b, c	h
SLDT = Store Local Descriptor Table Register					
To Register/Memory	00001111 00000000 mod 000 r/m	N/A	2/2	a	h

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
SMSW = Store Machine Status Word	00001111 00000001 mod100 r/m	2/2	2/2	b, c	h, l
STR = Store Task Register To Register/Memory	00001111 00000000 mod001 r/m	N/A	2/2	a	h
VERR = Verify Read Access Register/Memory	00001111 00000000 mod100 r/m	N/A	10/11	a	g, h, i, p
VERW = Verify Write Access	00001111 00000000 mod101 r/m	N/A	15/16	a	g, h, i, p

INSTRUCTION NOTES FOR TABLE 6-1
Notes a through c apply to 386 DX Real Address Mode only:

a. This is a Protected Mode instruction. Attempted execution in Real Mode will result in exception 6 (invalid opcode).

b. Exception 13 fault (general protection) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum CS, DS, ES, FS or GS limit, FFFFH. Exception 12 fault (stack segment limit violation or not present) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.

c. This instruction may be executed in Real Mode. In Real Mode, its purpose is primarily to initialize the CPU for Protected Mode.

Notes d through g apply to 386 DX Real Address Mode and 386 DX Protected Virtual Address Mode:

d. The 386 DX uses an early-out multiply algorithm. The actual number of clocks depends on the position of the most significant bit in the operand (multiplier).

Clock counts given are minimum to maximum. To calculate actual clocks use the following formula:

$$\text{Actual Clock} = \text{if } m < > 0 \text{ then } \max([\log_2 |m|], 3) + b \text{ clocks:}$$

$$\text{if } m = 0 \text{ then } 3 + b \text{ clocks}$$

In this formula, m is the multiplier, and

- b = 9 for register to register,
- b = 12 for memory to register,
- b = 10 for register with immediate to register,
- b = 11 for memory with immediate to register.

e. An exception may occur, depending on the value of the operand.

f. LOCK# is automatically asserted, regardless of the presence or absence of the LOCK# prefix.

g. LOCK# is asserted during descriptor table accesses.

Notes h through r apply to 386 DX Protected Virtual Address Mode only:

h. Exception 13 fault (general protection violation) will occur if the memory operand in CS, DS, ES, FS or GS cannot be used due to either a segment limit violation or access rights violation. If a stack limit is violated, an exception 12 (stack segment limit violation or not present) occurs.

i. For segment load operations, the CPL, RPL, and DPL must agree with the privilege rules to avoid an exception 13 fault (general protection violation). The segment's descriptor must indicate "present" or exception 11 (CS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 (stack segment limit violation or not present) occurs.

j. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert LOCK# to maintain descriptor integrity in multiprocessor systems.

k. JMP, CALL, INT, RET and IRET instructions referring to another code segment will cause an exception 13 (general protection violation) if an applicable privilege rule is violated.

l. An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).

m. An exception 13 fault occurs if CPL is greater than IOPL.

n. The IF bit of the flag register is not updated if CPL is greater than IOPL. The IOPL and VM fields of the flag register are updated only if CPL = 0.

o. The PE bit of the MSW (CR0) cannot be reset by this instruction. Use MOV into CR0 if desiring to reset the PE bit.

p. Any violation of privilege rules as applied to the selector operand does not cause a protection exception; rather, the zero flag is cleared.

q. If the coprocessor's memory operand violates a segment limit or segment access rights, an exception 13 fault (general protection exception) will occur before the ESC instruction is executed. An exception 12 fault (stack segment limit violation or not present) will occur if the stack limit is violated by the operand's starting address.

r. The destination of a JMP, CALL, INT, RET or IRET must be in the defined limit of a code segment or an exception 13 fault (general protection violation) will occur.

6.2 INSTRUCTION ENCODING

6.2.1 Overview

All instruction encodings are subsets of the general instruction format shown in Figure 6-1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the “mod r/m” byte and “scaled index” byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain

encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 6-1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 6-2 is a complete list of all fields appearing in the 386 DX instruction set. Further ahead, following Table 6-2, are detailed tables for each field.

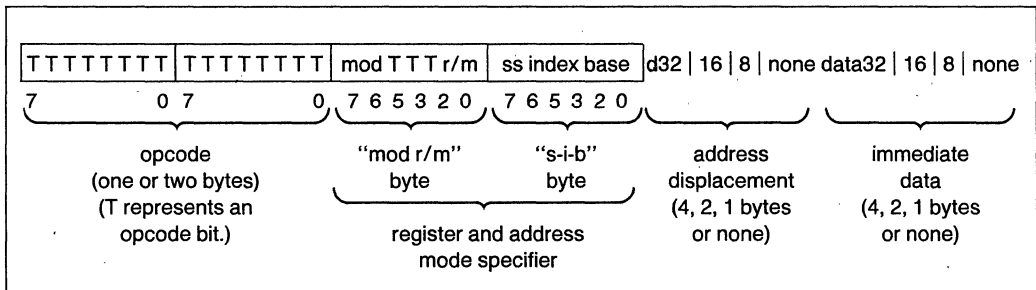


Figure 6-1. General instruction Format

Table 6-2. Fields within 386™ DX Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 Bits)	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field Must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
tttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

Note: Table 6-1 shows encoding of individual instructions.

6.2.2 32-Bit Extensions of the Instruction Set

With the 386 DX, the 8086/80186/80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 bits or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bits or 16 bits) for both operands and effective addresses when executing that code segment. In the Real Address Mode or Virtual 8086 Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the 386 DX when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the opcode bytes. The presence of the Operand Size Prefix and the Effective Address Prefix will toggle the operand size or the effective address size, respectively, to the value "opposite" from the Default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. As another example, if the default effective address size is 16 bits, presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.

These 32-bit extensions are available in all 386 DX modes, including the Real Address Mode or the Virtual 8086 Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

6.2.3 Encoding of Instruction Fields

Within the instruction are several fields indicating register selection, addressing mode and so on. The exact encodings of these fields are defined immediately ahead.

6.2.3.1 ENCODING OF OPERAND LENGTH (w) FIELD

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size During 16-Bit Data Operations	Operand Size During 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

6.2.3.2 ENCODING OF THE GENERAL REGISTER (reg) FIELD

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the "mod r/m" byte, or as the r/m field of the "mod r/m" byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-Bit Data Operations	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations:		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-Bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

6.2.3.3 ENCODING OF THE SEGMENT REGISTER (sreg) FIELD

The sreg field in certain instructions is a 2-bit field allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the 386 DX FS and GS segment registers to be specified.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

6.2.3.4 ENCODING OF ADDRESS MODE

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the "mod r/m" byte, and a second byte of addressing information, the "s-i-b" (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the "mod r/m" byte has r/m = 100 and mod = 00, 01 or 10. When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the "mod r/m" byte, also contains three bits (shown as TTT in Figure 6-1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the "mod r/m" byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the "mod r/m" byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following three pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Encoding of 16-bit Address Mode with “mod r/m” Byte

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	SS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by r/m During 16-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode with “mod r/m” byte (no “s-i-b” byte present):

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by reg or r/m during 16-Bit Data Operations:

mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by reg or r/m during 32-Bit Data Operations:

mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode (“mod r/m” byte and “s-i-b” byte present):

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

****IMPORTANT NOTE:**

When index field is 100, indicating “no index register,” then ss field **MUST** equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

NOTE:

Mod field in “mod r/m” byte; ss, index, base fields in “s-i-b” byte.

6.2.3.5 ENCODING OF OPERATION DIRECTION (d) FIELD

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory <- - Register "reg" Field Indicates Source Operand; "mod r/m" or "mod ss index base" Indicates Destination Operand
1	Register <- - Register/Memory "reg" Field Indicates Destination Operand; "mod r/m" or "mod ss index base" Indicates Source Operand

6.2.3.6 ENCODING OF SIGN-EXTEND (s) FIELD

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16 32
0	None	None
1	Sign-Extend Data8 to Fill 16-Bit or 32-Bit Destination	None

6.2.3.7 ENCODING OF CONDITIONAL TEST (ttn) FIELD

For the conditional instructions (conditional jumps and set on condition), ttn is encoded with n indicating to use the condition (n=0) or its negation (n=1), and tti giving the condition to test.

Mnemonic	Condition	ttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

6.2.3.8 ENCODING OF CONTROL OR DEBUG OR TEST REGISTER (eee) FIELD

For the loading and storing of the Control, Debug and Test registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3
Do not use any other encoding	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

When Interpreted as Test Register Field

eee Code	Reg Name
110	TR6
111	TR7
Do not use any other encoding	

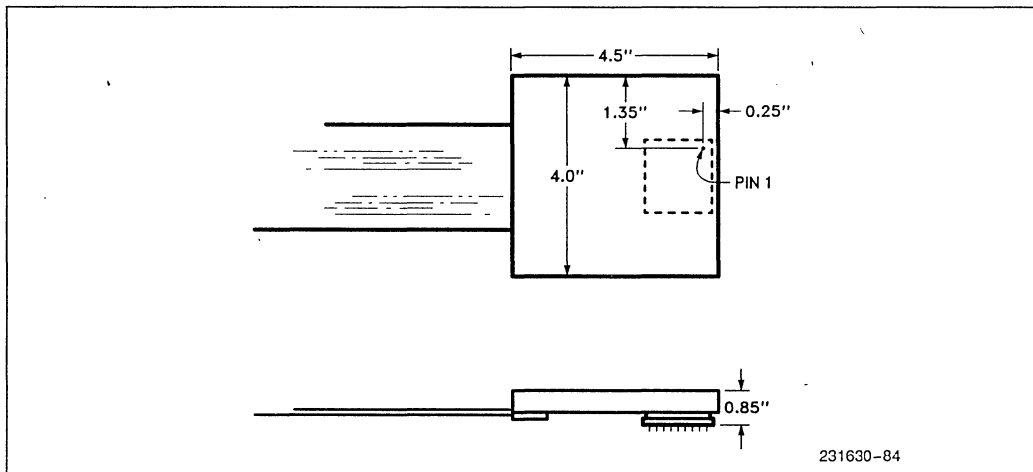


Figure 7-1. Processor Module Dimensions

7. DESIGNING FOR ICETM-386 DX EMULATOR USE

The 386 DX in-circuit emulator products are ICE-386 DX 25 MHz or 33 MHz (both referred to as ICE-386 DX emulator). The ICE-386 DX emulator probe module has several electrical and mechanical characteristics that should be taken into consideration when designing the hardware.

Capacitive loading: The ICE-386 DX emulator adds up to 25 pF to each line.

Drive requirement: The ICE-386 DX emulator adds one standard TTL load on the CLK2 line, up to one advanced low-power Schottky TTL load per control signal line, and one advanced low-power Schottky TTL load per address, byte enable, and data line. These loads are within the probe module and are driven by the probe's 386 DX component, which has standard drive and loading capability listed in the A.C. and D.C. Specification Tables in Sections 9.4 and 9.5.

Power requirement: For noise immunity the ICE-386 DX emulator probe is powered by the user system. This high-speed probe circuitry draws up to 1.5A plus the maximum I_{CC} from the user 386 DX component socket.

386 DX location and orientation: The ICE-386 DX processor module, target-adaptor cable (which does not exist for the ICE-386 DX 33 MHz emulator), and the isolation board used for extra electrical buffering of the emulator initially, require clearance as illustrated in Figures 7-1 and 7-2.

Interface Board and CLK2 speed reduction: When the ICE-386 DX emulator probe is first attached to an unverified user system, the interface board helps the ICE-386 DX emulator function in user systems with bus faults (shorted signals, etc.). After electrical verification it may be removed. Only when the interface board is installed, the user system must have a reduced CLK2 frequency of 25 MHz maximum.

Cache coherence: The ICE-386 DX emulator loads user memory by performing 386 DX component write cycles. Note that if the user system is not designed to update or invalidate its cache (if it has a cache) upon processor writes to memory, the cache could contain stale instruction code and/or data. For best use of the ICE-386 DX emulator, the user should consider designing the cache (if any) to update itself automatically when processor writes occur, or find another method of maintaining cache data coherence with main user memory.

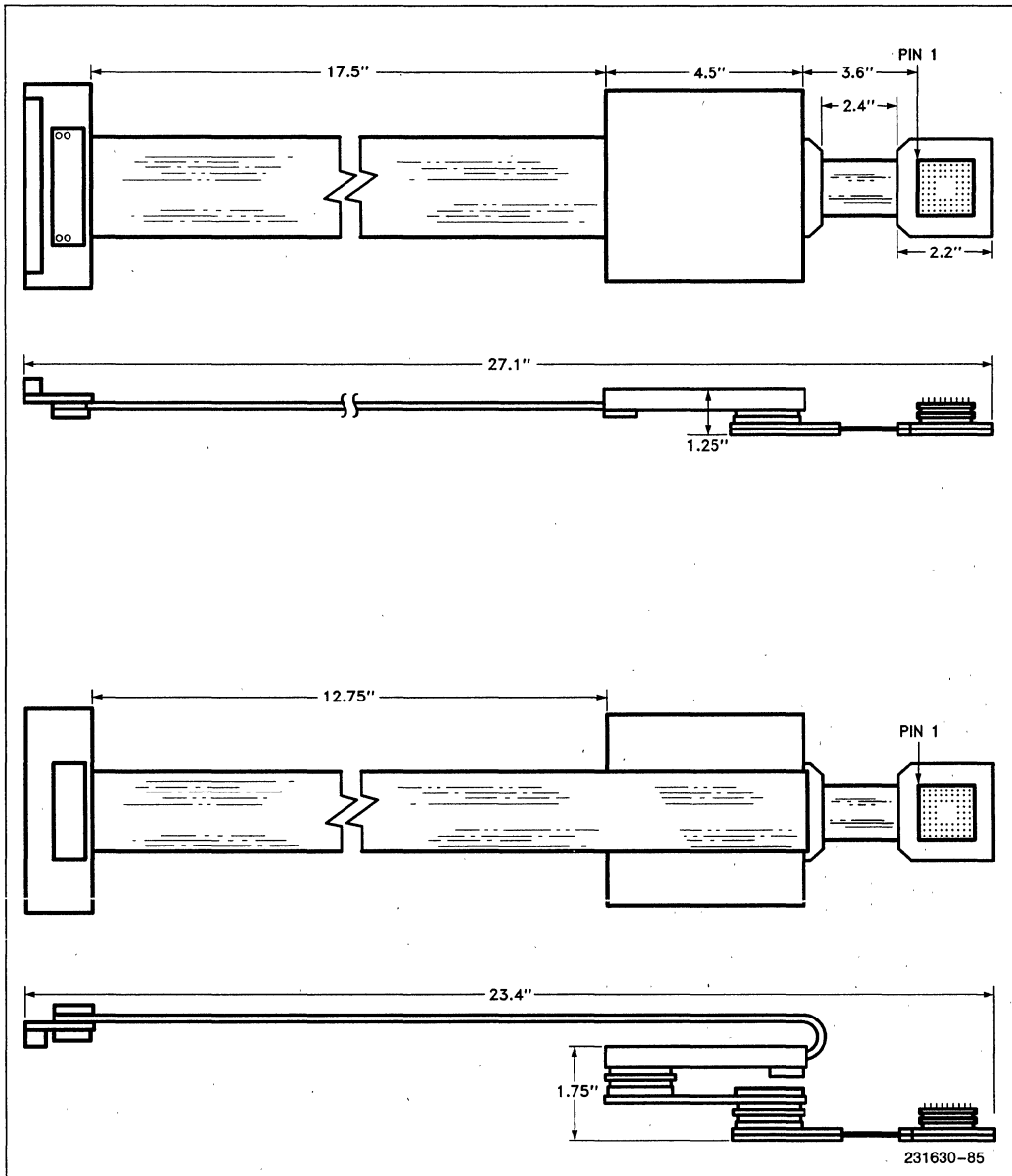


Figure 7-2. Processor Module, Target-Adapter Cable, and Isolation Board Dimensions

8. MECHANICAL DATA

8.1 INTRODUCTION

In this section, the physical packaging and its connections are described in detail.

8.2 PACKAGE DIMENSIONS AND MOUNTING

The initial 386 DX package is a 132-pin ceramic pin grid array (PGA). Pins of this package are arranged 0.100 inch (2.54mm) center-to-center, in a 14 x 14 matrix, three rows around.

A wide variety of available sockets allow low insertion force or zero insertion force mountings, and a choice of terminals such as soldertail, surface mount, or wire wrap. Several applicable sockets are listed in Table 8.1.

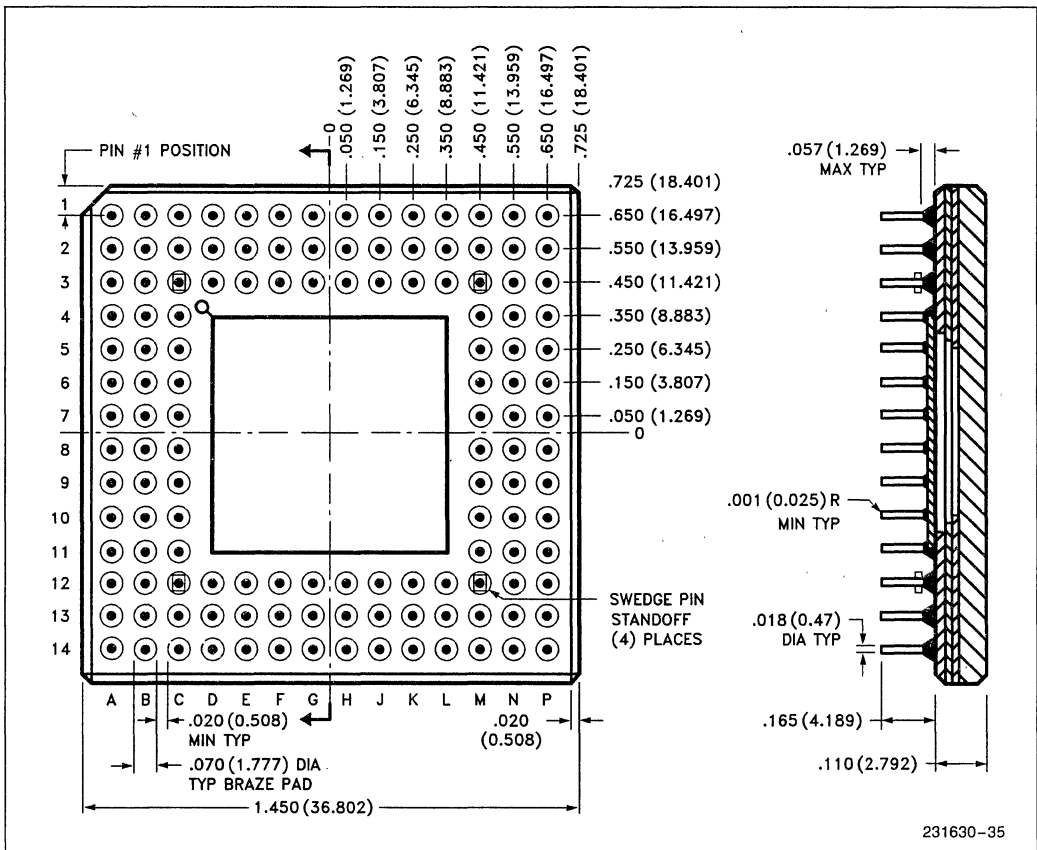


Figure 8.1. 132-Pin Ceramic PGA Package Dimensions

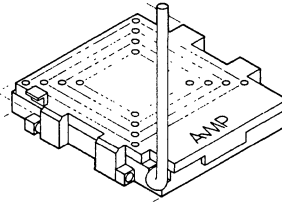
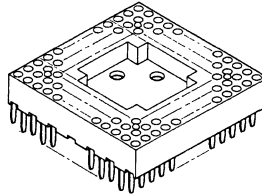
Table 8.1. Several Socket Options for 132-Pin PGA

- * Low insertion force (LIF) soldertail 55274-1
- * Amp tests indicate 50% reduction in insertion force compared to machined sockets

Other socket options

- * Zero insertion force (ZIF) soldertail 55583-1
- * Zero insertion force (ZIF) Burn-in version 55573-2

Amp Incorporated
 (Harrisburg, PA 17105 U.S.A.)
 Phone 717-564-0100



231630-45

Cam handle locks in low profile position when substrate is installed (handle UP for open and DOWN for closed positions)

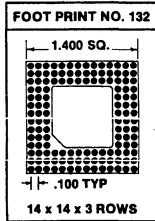
courtesy Amp Incorporated

Peel-A-Way™ Mylar and Kapton Socket Terminal Carriers

- * Low insertion force surface mount CS132-37TG
- * Low insertion force soldertail CS132-01TG
- * Low insertion force wire-wrap CS132-02TG (two level) CS132-03TG (three-level)
- * Low insertion force press-fit CS132-05TG

Peel-A-Way Carrier No. 132: Kapton Carrier is KS132 Mylar Carrier is MS132

Molded Plastic Body KS132 is shown below:



231630-46

Advanced Interconnections
 (5 Division Street
 Warwick, RI 02818 U.S.A.)
 Phone 401-935-0495)

SOLDER TAIL -01	LOW PROFILE -04	PRESS FIT -05
MILLIMETER INCH		STG. HOLE P.T.H. PRIMARY .020 FINISHED .022 ± .002
WIRE WRAP -02/-03	PEEL-A-WAY	SURFACE MOUNTING -37

231630-47

courtesy Advanced Interconnections
 (Peel-A-Way Terminal Carriers
 U.S. Patent No. 4442938)

Table 8.1. Several Socket Options for 132-Pin PGA (Continued)

**PIN GRID ARRAY
DECOUPLING SOCKETS**

- Low insertion force soldertail
0.125 length PGD-005-1A1
Finish: Term/Contact Tin-
Lead/Gold
- Low insertion force soldertail
0.180 length PGD-005-1B1
Finish: Term/Contact: Tin-
Lead/Gold
- Low insertion 3 level Wire/
Wrap PGD-005-1C1 Finish:
Term/Contact Tin-Lead/Gold

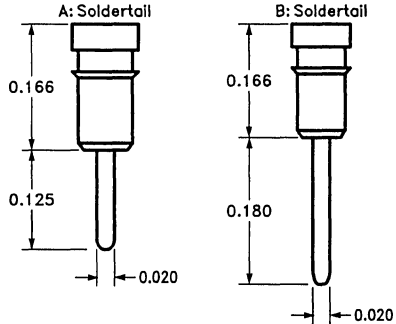
Includes 0.10 μ F & 1.0 μ F
Decoupling Capacitors

VisinPak Kapton Carrier

PKC Series

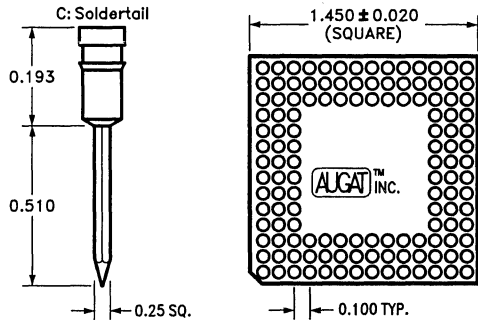
Pin Grid Array

PGM (Plastic) or PPS
(Glass Epoxy) Series



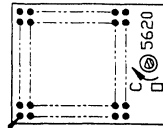
AUGAT INC.

33 Perry Ave., P.O. Box 779 Attleboro, MA 02703
TECHNICAL INFORMATION: (508) 222-2202
CUSTOMER SERVICE: (508) 699-9800



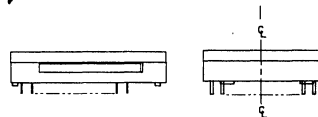
231630-86

- Low insertion force socket soldertail
(for production use)
2XX-6576-00-3308 (new style)
2XX-6003-00-3302 (older style)
- Zero insertion force soldertail
(for test and burn-in use)
2XX-6568-00-3302



Textool Products

Electronic Products Division/3M
(1410 West Pioneer Drive
Irving, Texas 75601 U.S.A.
Phone 214-259-2676)



courtesy Textool Products/3M

231630-48

8.3 PACKAGE THERMAL SPECIFICATION

The 386 DX is specified for operation when case temperature is within the range of 0°C–85°C. The case temperature may be measured in any environment, to determine whether the 386 DX is within specified operating range.

The PGA case temperature should be measured at the center of the top surface opposite the pins, as in Figure 8.2.

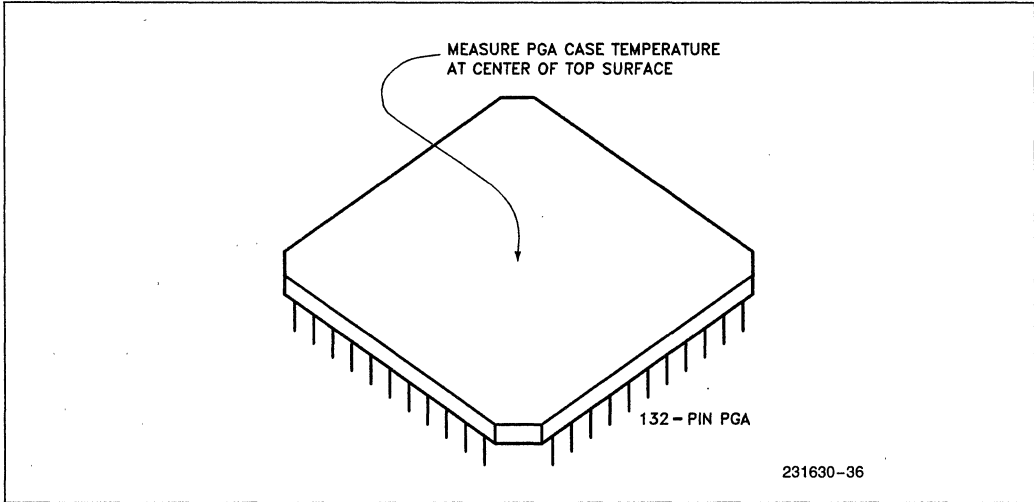
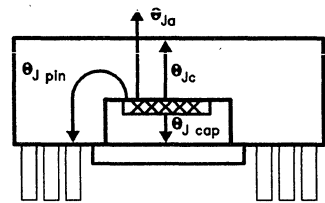


Figure 8.2. Measuring 386™ DX PGA Case Temperature

Table 8.2. 386™ DX PGA Package Thermal Characteristics

Parameter	Thermal Resistance — °C/Watt						
	Airflow — ft./min (m/sec)						
	0 (0)	50 (0.25)	100 (0.50)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
θ Junction-to-Case (case measured as Fig. 6-4)	2	2	2	2	2	2	2
θ Case-to-Ambient (no heatsink)	19	18	17	15	12	10	9
θ Case-to-Ambient (with omnidirectional heatsink)	16	15	14	12	9	7	6
θ Case-to-Ambient (with unidirectional heatsink)	15	14	13	11	8	6	5



NOTES:

- Table 8.2 applies to 386™ DX PGA plugged into socket or soldered directly into board.
- $\theta_{JA} = \theta_{JC} + \theta_{CA}$.

- $\theta_{J-CAP} = 4^\circ\text{C/w}$ (approx.)
 $\theta_{J-PIN} = 4^\circ\text{C/w}$ (inner pins) (approx.)
 $\theta_{J-PIN} = 8^\circ\text{C/w}$ (outer pins) (approx.)

9. ELECTRICAL DATA

9.1 INTRODUCTION

The following sections describe recommended electrical connections for the 386 DX, and its electrical specifications.

9.2 POWER AND GROUNDING

9.2.1 Power Connections

The 386 DX is implemented in CHMOS III and CHMOS IV technology and has modest power requirements. However, its high clock frequency and 72 output buffers (address, data, control, and HLDA) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 20 V_{CC} and 21 V_{SS} pins separately feed functional units of the 386 DX.

Power and ground connections must be made to all external V_{CC} and GND pins of the 386 DX. On the circuit board, all V_{CC} pins must be connected on a V_{CC} plane. All V_{SS} pins must be likewise connected on a GND plane.

9.2.2 Power Decoupling Recommendations

Liberal decoupling capacitance should be placed near the 386 DX. The 386 DX driving its 32-bit parallel address and data buses at high frequencies can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the 386 DX and

decoupling capacitors as much as possible. Capacitors specifically for PGA packages are also commercially available, for the lowest possible inductance.

9.2.3 Resistor Recommendations

The ERROR # and BUSY # inputs have resistor pullups of approximately 20 K Ω built-in to the 386 DX to keep these signals negated when no 387 DX co-processor is present in the system (or temporarily removed from its socket). The BS16# input also has an internal pullup resistor of approximately 20 K Ω , and the PEREQ input has an internal pulldown resistor of approximately 20 K Ω .

In typical designs, the external pullup resistors shown in Table 9-1 are recommended. However, a particular design may have reason to adjust the resistor values recommended here, or alter the use of pullup resistors in other ways.

9.2.4 Other Connection Recommendations

For reliable operation, always connect unused inputs to an appropriate signal level. N.C. pins should always remain unconnected.

Particularly when not using interrupts or bus hold, (as when first prototyping, perhaps) prevent any chance of spurious activity by connecting these associated inputs to GND:

Pin	Signal
B7	INTR
B8	NMI
D14	HOLD

If not using address pipelining, pullup D13 NA# to V_{CC} .

If not using 16-bit bus size, pullup C14 BS16# to V_{CC} .

Pullups in the range of 20 K Ω are recommended.

Table 9-1. Recommended Resistor Pullups to V_{CC}

Pin and Signal	Pullup Value	Purpose
E14 ADS#	20 K Ω \pm 10%	Lightly Pull ADS# Negated During 386™ DX Hold Acknowledge States
C10 LOCK#	20 K Ω \pm 10%	Lightly Pull LOCK# Negated During 386™ DX Hold Acknowledge States

9.3 MAXIMUM RATINGS

Table 9-2. Maximum Ratings

Parameter	386™ DX 20, 25, 33 MHz Maximum Rating
Storage Temperature	-65°C to +150°C
Case Temperature Under Bias	-65°C to +110°C
Supply Voltage with Respect to V _{SS}	-0.5V to +6.5V
Voltage on Other Pins	-0.5V to V _{CC} + 0.5V

Table 9-2 is a stress rating only, and functional operation at the maximums is not guaranteed. Functional operating conditions are given in **9.4 D.C. Specifications** and **9.5 A.C. Specifications**.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the 386 DX contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

9.4 D.C. SPECIFICATIONS

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to 85°C

Table 9-3. 386™ DX D.C. Characteristics

Symbol	Parameter	386™ DX 20 MHz, 25 MHz, 33 MHz		Unit	Test Conditions
		Min	Max		
V _{IL}	Input Low Voltage	-0.3	0.8	V	(Note 1)
V _{IH}	Input High Voltage	2.0	V _{CC} + 0.3	V	
V _{ILC}	CLK2 Input Low Voltage	-0.3	0.8	V	(Note 1)
V _{IHC}	CLK2 Input High Voltage 20 MHz 25 MHz and 33 MHz	V _{CC} - 0.8	V _{CC} + 0.3	V	
		3.7	V _{CC} + 0.3	V	
V _{OL}	Output Low Voltage I _{OL} = 4 mA: A2-A31, D0-D31 I _{OL} = 5 mA: BE0#-BE3#, W/R#, D/C#, M/IO#, LOCK#, ADS#, HLDA		0.45	V	
			0.45	V	
V _{OH}	Output High Voltage I _{OH} = 1 mA: A2-A31, D0-D31 I _{OH} = 0.9 mA: BE0#-BE3#, W/R#, D/C#, M/IO#, LOCK#, ADS#, HLDA	2.4		V	
		2.4		V	
I _{LI}	Input Leakage Current (For All Pins except BS16#, PEREQ, BUSY#, and ERROR#)		± 15	μA	0V ≤ V _{IN} ≤ V _{CC}
I _{IH}	Input Leakage Current (PEREQ Pin)		200	μA	V _{IH} = 2.4V (Note 2)
I _{IL}	Input Leakage Current (BS16#, BUSY#, and ERROR# Pins)		-400	μA	V _{IL} = 0.45 (Note 3)
I _{LO}	Output Leakage Current		± 15	μA	0.45V ≤ V _{OUT} ≤ V _{CC}
I _{CC}	Supply Current CLK2 = 40 MHz: with 20 MHz 386™ DX CLK2 = 50 MHz: with 25 MHz 386™ DX CLK2 = 66 MHz: with 33 MHz 386™ DX		500	mA	I _{CC} Typ. = 460 mA
			550	mA	I _{CC} Typ. = 500 mA
			550	mA	I _{CC} Typ. = 400 mA
C _{IN}	Input or I/O Capacitance		10	pF	F _C = 1 MHz (Note 4)
C _{OUT}	Output Capacitance		12	pF	F _C = 1 MHz (Note 4)
C _{CLK}	CLK2 Capacitance		20	pF	F _C = 1 MHz (Note 4)

NOTES:

1. The min value, -0.3, is not 100% tested.
2. PEREQ input has an internal pulldown resistor.
3. BS16#, BUSY# and ERROR# inputs each have an internal pullup resistor.
4. Not 100% tested.

9.5 A.C. SPECIFICATIONS

9.5.1 A.C. Spec Definitions

The A.C. specifications, given in Tables 9-4, 9-5, and 9-6, consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the CLK2 rising edge crossing the 2.0V level.

A.C. spec measurement is defined by Figure 9-1. Inputs must be driven to the voltage levels indicated by Figure 9-1 when A.C. specifications are measured. 386 DX output delays are specified with minimum and maximum limits, measured as shown. The minimum 386 DX delay times are hold times

provided to external circuitry. 386 DX input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 386 DX operation.

Outputs NA#, W/R#, D/C#, M/IO#, LOCK#, BE0#-BE3#, A2-A31 and HLDA only change at the beginning of phase one. D0-D31 (write cycles) only change at the beginning of phase two. The READY#, HOLD, BUSY#, ERROR#, PEREQ and D0-D31 (read cycles) inputs are sampled at the beginning of phase one. The NA#, BS16#, INTR and NMI inputs are sampled at the beginning of phase two.

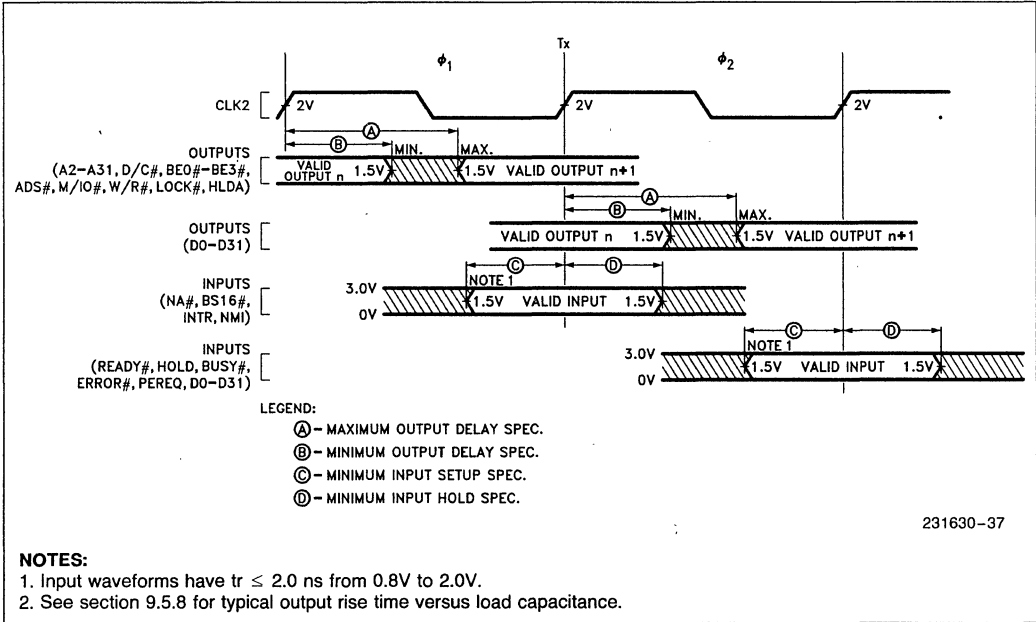


Figure 9-1. Drive Levels and Measurement Points for A.C. Specifications

9.5.2 A.C. Specification Tables

 Functional Operating Range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$
Table 9-4. 33 MHz 386™ DX A.C. Characteristics

Symbol	Parameter	33 MHz 386™ DX		Unit	Ref. Fig.	Notes
		Min	Max			
	Operating Frequency	8	33.3	MHz		Half of CLK2 Frequency
t1	CLK2 Period	15.0	62.5	ns	9-3	
t2a	CLK2 High Time	6.25		ns	9-3	at 2V
t2b	CLK2 High Time	4.5		ns	9-3	at 3.7V
t3a	CLK2 Low Time	6.25		ns	9-3	at 2V
t3b	CLK2 Low Time	4.5		ns	9-3	at 0.8V
t4	CLK2 Fall Time		4	ns	9-3	3.7V to 0.8V (Note 3)
t5	CLK2 Rise Time		4	ns	9-3	0.8V to 3.7V (Note 3)
t6	A2–A31 Valid Delay	4	15	ns	9-5	$C_L = 50$ pF
t7	A2–A31 Float Delay	4	20	ns	9-6	(Note 1)
t8	BE0# –BE3#, LOCK# Valid Delay	4	15	ns	9-5	$C_L = 50$ pF
t9	BE0# –BE3#, LOCK# Float Delay	4	20	ns	9-6	(Note 1)
t10	W/R#, M/IO#, D/C#, Valid Delay	4	15	ns	9-5	$C_L = 50$ pF
t10a	ADS# Valid Delay	4	14.5	ns	9-5	$C_L = 50$ pF
t11	W/R#, M/IO#, D/C#, ADS# Float Delay	4	20	ns	9-6	(Note 1)
t12	D0–D31 Write Data Valid Delay	7	24	ns	9-5a	$C_L = 50$ pF, (Note 4)
t12a	D0–D31 Write Data Hold Time	2			9-5b	$C_L = 50$ pF
t13	D0–D31 Float Delay	4	17	ns	9-6	(Note 1)
t14	HLDA Valid Delay	4	20	ns	9-6	$C_L = 50$ pF
t15	NA# Setup Time	5		ns	9-4	
t16	NA# Hold Time	2		ns	9-4	
t17	BS16# Setup Time	5		ns	9-4	
t18	BS16# Hold Time	2		ns	9-4	
t19	READY# Setup Time	7		ns	9-4	
t20	READY# Hold Time	4		ns	9-4	

9.5.2 A.C. Specification Tables (Continued)

 Functional Operating Range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$
Table 9-4. 33 MHz 386™ DX A.C. Characteristics (Continued)

Symbol	Parameter	33 MHz 386™ DX		Unit	Ref. Fig.	Notes
		Min	Max			
t21	D0–D31 Read Setup Time	5		ns	9-4	
t22	D0–D31 Read Hold Time	3		ns	9-4	
t23	HOLD Setup Time	11		ns	9-4	
t24	HOLD Hold Time	2		ns	9-4	
t25	RESET Setup Time	5		ns	9-7	
t26	RESET Hold Time	2		ns	9-7	
t27	NMI, INTR Setup Time	5		ns	9-4	(Note 2)
t28	NMI, INTR Hold Time	5		ns	9-4	(Note 2)
t29	PEREQ, ERROR #, BUSY # Setup Time	5		ns	9-4	(Note 2) [±]
t30	PEREQ, ERROR #, BUSY # Hold Time	4		ns	9-4	(Note 2)

NOTES:

1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.
3. Rise and fall times are not tested.
4. Min. time not 100% tested.

9.5.2 A.C. Specification Tables (Continued)

 Functional Operating Range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$
Table 9-5. 25 MHz 386™ DX A.C. Characteristics

Symbol	Parameter	25 MHz 386™ DX		Unit	Ref. Fig.	Notes
		Min	Max			
	Operating Frequency	4	25	MHz		Half of CLK2 Frequency
t1	CLK2 Period	20	125	ns	9-3	
t2a	CLK2 High Time	7		ns	9-3	at 2V
t2b	CLK2 High Time	4		ns	9-3	at 3.7V
t3a	CLK2 Low Time	7		ns	9-3	at 2V
t3b	CLK2 Low Time	5		ns	9-3	at 0.8V
t4	CLK2 Fall Time		7	ns	9-3	3.7V to 0.8V
t5	CLK2 Rise Time		7	ns	9-3	0.8V to 3.7V
t6	A2–A31 Valid Delay	4	21	ns	9-5	$C_L = 50$ pF
t7	A2–A31 Float Delay	4	30	ns	9-6	(Note 1)
t8	BE0#–BE3# Valid Delay	4	24	ns	9-5	$C_L = 50$ pF
t8a	LOCK# Valid Delay	4	21	ns	9-5	$C_L = 50$ pF
t9	BE0#–BE3#, LOCK# Float Delay	4	30	ns	9-6	(Note 1)
t10	W/R#, M/IO#, D/C#, ADS# Valid Delay	4	21	ns	9-5	$C_L = 50$ pF
t11	W/R#, M/IO#, D/C#, ADS# Float Delay	4	30	ns	9-6	(Note 1)
t12	D0–D31 Write Data Valid Delay	7	27	ns	9-5a	$C_L = 50$ pF
t12a	D0–D31 Write Data Hold Time	2			9-5b	$C_L = 50$ pF
t13	D0–D31 Float Delay	4	22	ns	9-6	(Note 1)
t14	HLDA Valid Delay	4	22	ns	9-6	$C_L = 50$ pF
t15	NA# Setup Time	7		ns	9-4	
t16	NA# Hold Time	3		ns	9-4	
t17	BS16# Setup Time	7		ns	9-4	
t18	BS16# Hold Time	3		ns	9-4	
t19	READY# Setup Time	9		ns	9-4	
t20	READY# Hold Time	4		ns	9-4	



9.5.2 A.C. Specification Tables (Continued)

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 9-5. 25 MHz 386™ DX A.C. Characteristics (Continued)

Symbol	Parameter	25 MHz 386™ DX		Unit	Ref. Fig.	Notes
		Min	Max			
t21	D0–D31 Read Setup Time	7		ns	9-4	
t22	D0–D31 Read Hold Time	5		ns	9-4	
t23	HOLD Setup Time	15		ns	9-4	
t24	HOLD Hold Time	3		ns	9-4	
t25	RESET Setup Time	10		ns	9-7	
t26	RESET Hold Time	3		ns	9-7	
t27	NMI, INTR Setup Time	6		ns	9-4	(Note 2)
t28	NMI, INTR Hold Time	6		ns	9-4	(Note 2)
t29	PEREQ, ERROR #, BUSY # Setup Time	6		ns	9-4	(Note 2)
t30	PEREQ, ERROR #, BUSY # Hold Time	5		ns	9-4	(Notes 2, 3)

NOTES:

1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.

3.

	Symbol	Parameter	Min
T _C = 0°C	t30	PEREQ, ERROR #, BUSY # Hold Time	4
T _C = +85°C	t30	PEREQ, ERROR #, BUSY # Hold Time	5

9.5.2 A.C. Specification Tables (Continued)

 Functional Operating Range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$
Table 9.6. 20 MHz 386™ DX A.C. Characteristics

Symbol	Parameter	20 MHz 386™ DX		Unit	Ref. Fig.	Notes
		Min	Max			
	Operating Frequency	4	20	MHz		Half of CLK2 Frequency
t_1	CLK2 Period	25	125	ns	9-3	
t_{2a}	CLK2 High Time	8		ns	9-3	at 2V
t_{2b}	CLK2 High Time	5		ns	9-3	at ($V_{CC} - 0.8V$)
t_{3a}	CLK2 Low Time	8		ns	9-3	at 2V
t_{3b}	CLK2 Low Time	6		ns	9-3	at 0.8V
t_4	CLK2 Fall Time		8	ns	9-3	($V_{CC} - 0.8V$) to 0.8V
t_5	CLK2 Rise Time		8	ns	9-3	0.8V to ($V_{CC} - 0.8V$)
t_6	A2–A31 Valid Delay	4	30	ns	9-5	$C_L = 120$ pF
t_7	A2–A31 Float Delay	4	32	ns	9-6	(Note 1)
t_8	BE0#–BE3#, LOCK# Valid Delay	4	30	ns	9-5	$C_L = 75$ pF
t_9	BE0#–BE3#, LOCK# Float Delay	4	32	ns	9-6	(Note 1)
t_{10}	W/R#, M/IO#, D/C#, ADS# Valid Delay	6	28	ns	9-5	$C_L = 75$ pF
t_{11}	W/R#, M/IO#, D/C#, ADS# Float Delay	6	30	ns	9-6	(Note 1)
t_{12}	D0–D31 Write Data Valid Delay	4	38	ns	9-5c	$C_L = 120$ pF
t_{13}	D0–D31 Float Delay	4	27	ns	9-6	(Note 1)
t_{14}	HLDA Valid Delay	6	28	ns	9-6	$C_L = 75$ pF
t_{15}	NA# Setup Time	9		ns	9-4	
t_{16}	NA# Hold Time	14		ns	9-4	
t_{17}	BS16# Setup Time	13		ns	9-4	
t_{18}	BS16# Hold Time	21		ns	9-4	
t_{19}	READY# Setup Time	12		ns	9-4	
t_{20}	READY# Hold Time	4		ns	9-4	
t_{21}	D0–D31 Read Setup Time	11		ns	9-4	
t_{22}	D0–D31 Read Hold Time	6		ns	9-4	
t_{23}	HOLD Setup Time	17		ns	9-4	
t_{24}	HOLD Hold Time	5		ns	9-4	
t_{25}	RESET Setup Time	12		ns	9-7	

9.5.2 A.C. Specification Tables (Continued)

 Functional Operating Range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$
Table 9-6. 20 MHz 386™ DX A.C. Characteristics (Continued)

Symbol	Parameter	20 MHz 386™ DX		Unit	Ref. Fig.	Notes
		Min	Max			
t ₂₆	RESET Hold Time	4		ns	9-7	
t ₂₇	NMI, INTR Setup Time	16		ns	9-4	(Note 2)
t ₂₈	NMI, INTR Hold Time	16		ns	9-4	(Note 2)
t ₂₉	PEREQ, ERROR #, BUSY # Setup Time	14		ns	9-4	(Note 2)
t ₃₀	PEREQ, ERROR #, BUSY # Hold Time	5		ns	9-4	(Note 2)

NOTES:

1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.

9.5.3 A.C. Test Loads

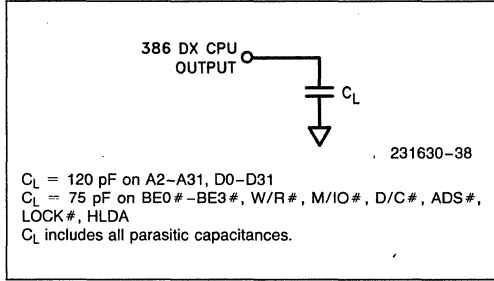


Figure 9-2. A.C. Test Load

9.5.4 A.C. Timing Waveforms

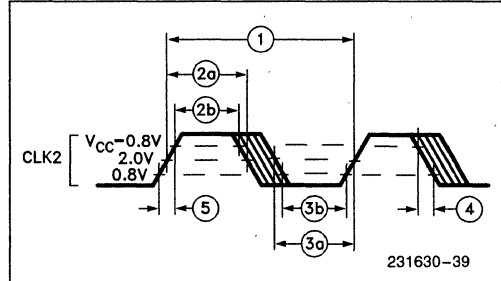


Figure 9-3. CLK2 Timing

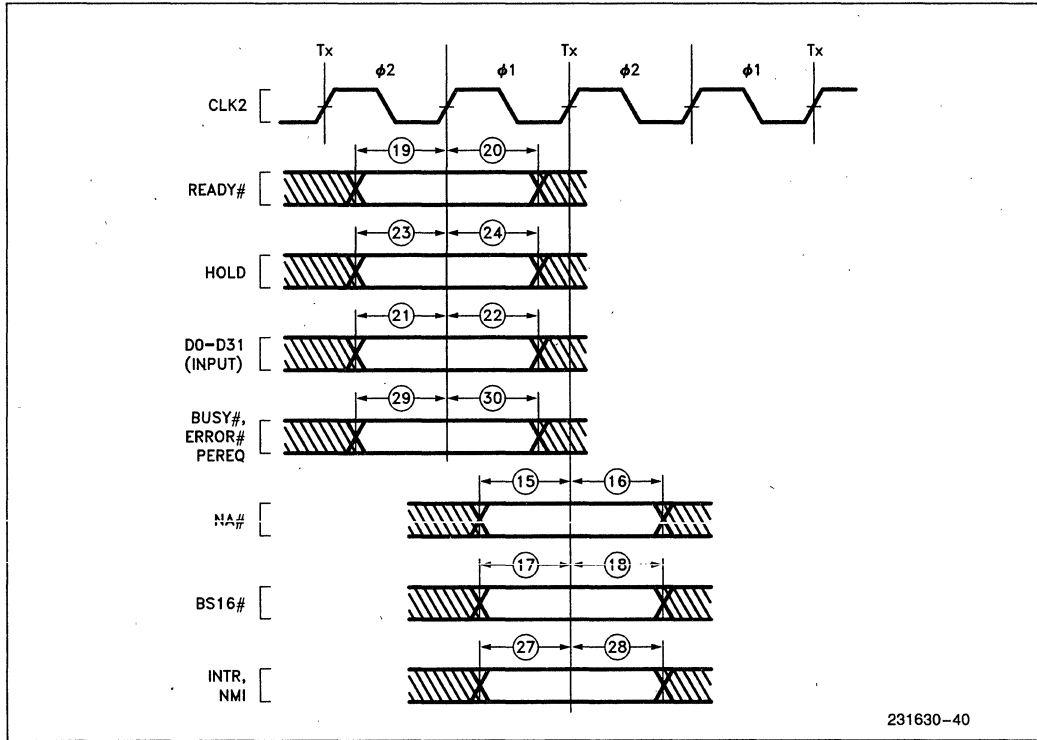


Figure 9-4. Input Setup and Hold Timing

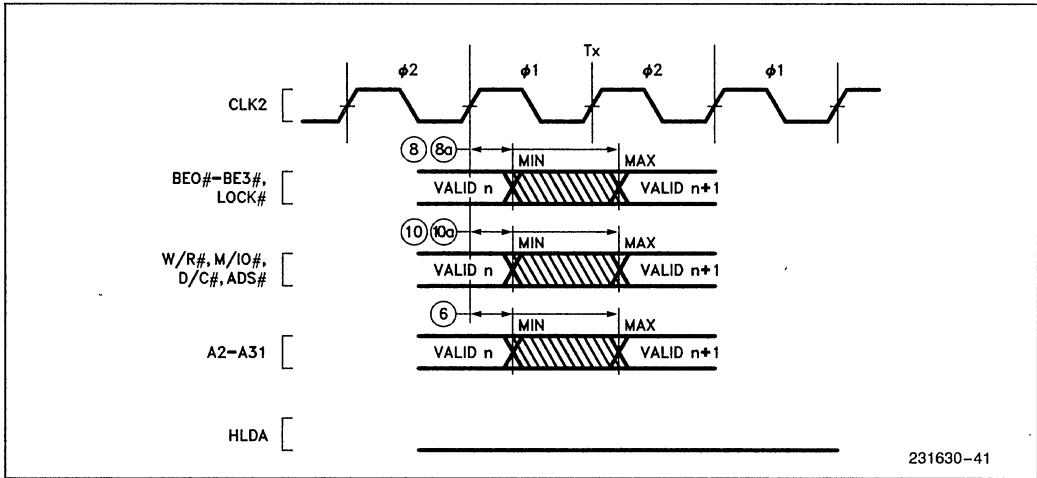


Figure 9-5. Output Valid Delay Timing

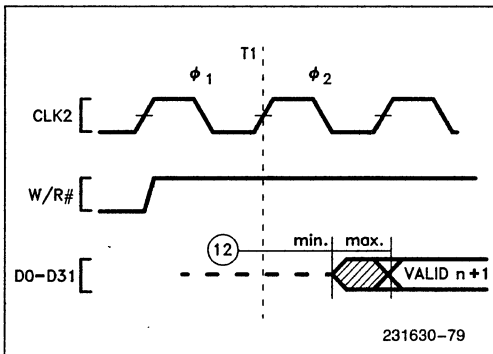


Figure 9-5a. Write Data Valid Delay Timing (25 MHz, 33 MHz)

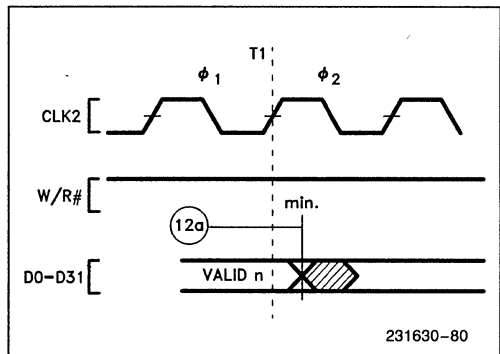


Figure 9-5b. Write Data Hold Timing (25 MHz, 33 MHz)

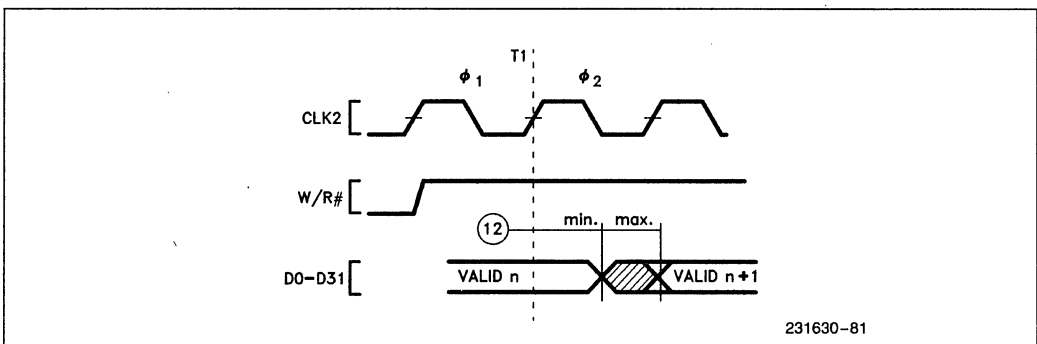
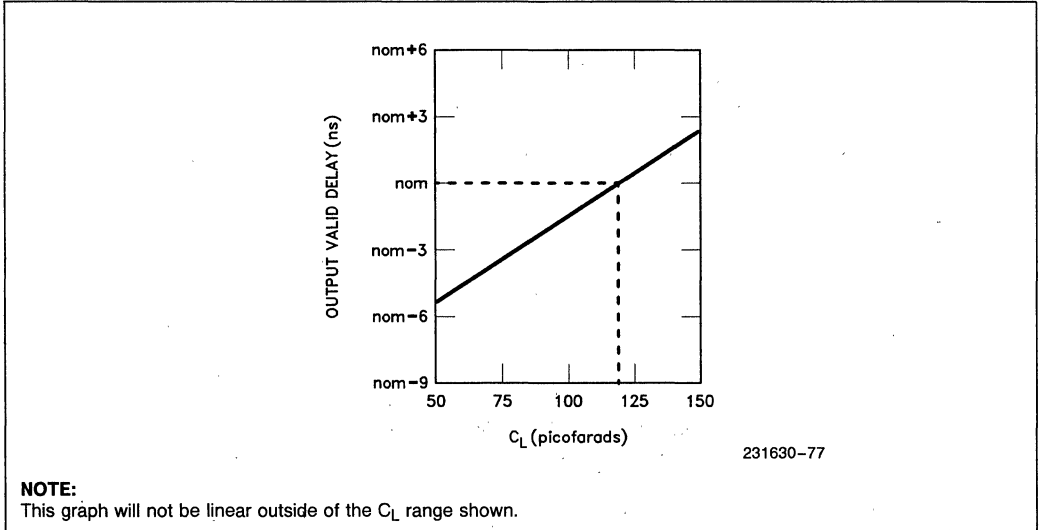
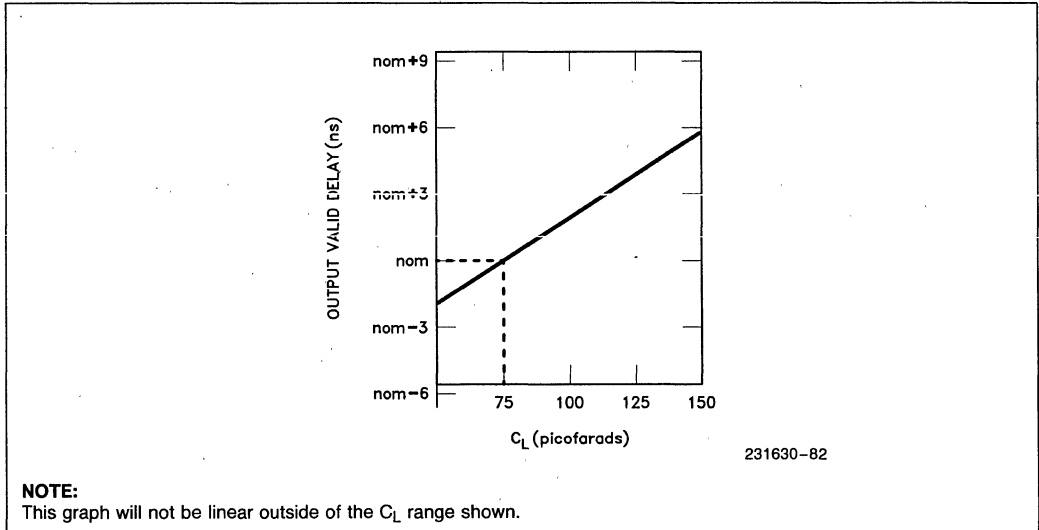


Figure 9-5c. Write Data Valid Delay Timing (20 MHz)

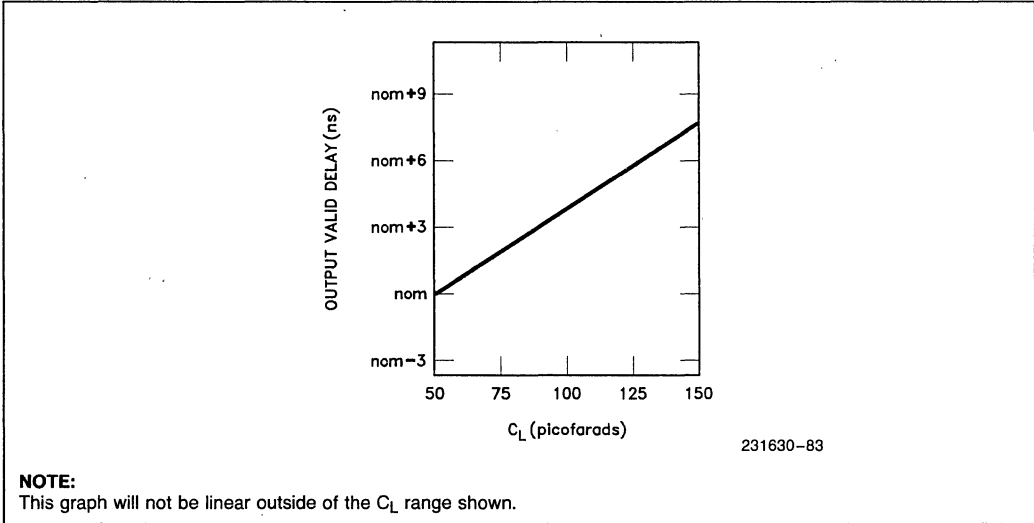
9.5.5 Typical Output Valid Delay Versus Load Capacitance at Maximum Operating Temperature ($C_L = 120 \text{ pF}$)



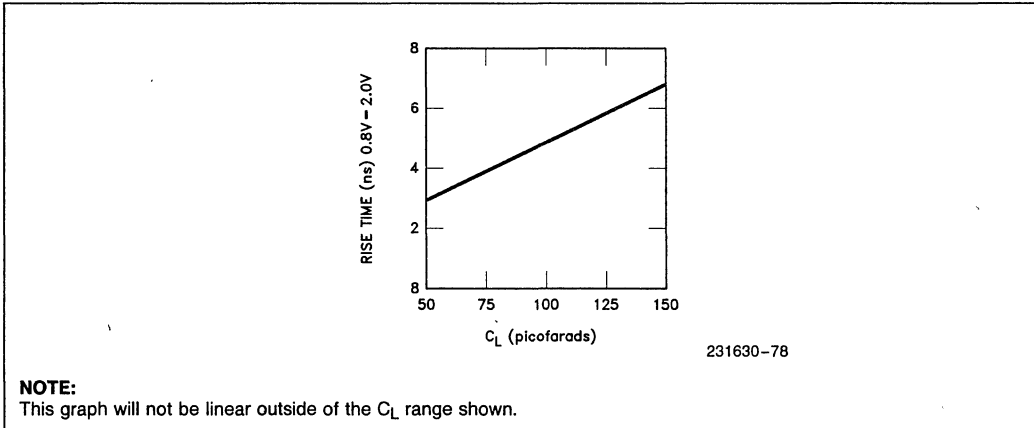
9.5.6 Typical Output Valid Delay Versus Load Capacitance at Maximum Operating Temperature ($C_L = 75 \text{ pF}$)



9.5.7 Typical Output Valid Delay Versus Load Capacitance at Maximum Operating Temperature ($C_L = 50$ pF)



9.5.8 Typical Output Rise Time Versus Load Capacitance at Maximum Operating Temperature



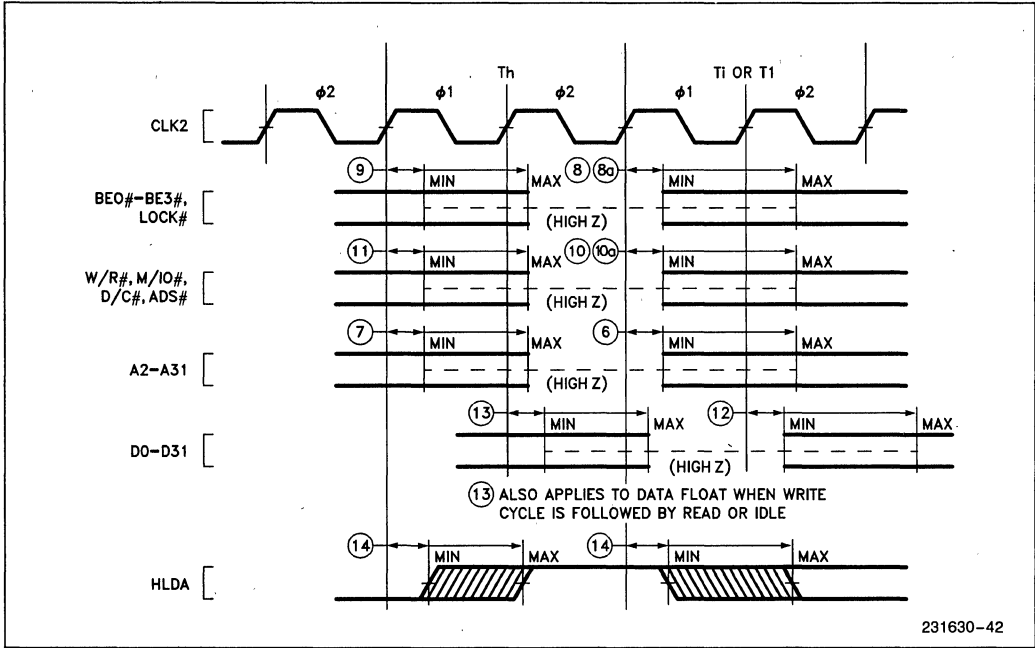
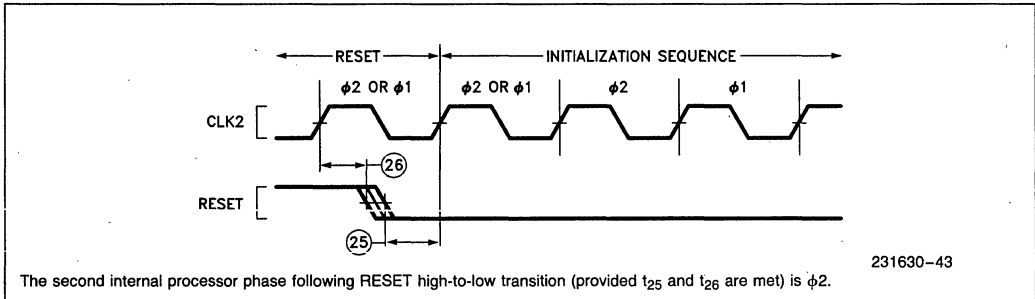


Figure 9-6. Output Float Delay and HLDA Valid Delay Timing



The second internal processor phase following RESET high-to-low transition (provided t_{25} and t_{26} are met) is ϕ_2 .

Figure 9-7. RESET Setup and Hold Timing, and Internal Phase

10.0 Revision History

This 386 DX data sheet, version -005, contains updates and improvements to previous versions. A revision summary is listed here for your convenience.

The sections significantly revised since version -001 are:

2.9.6	Sequence of exception checking table added.
2.9.7	Instruction restart revised.
2.11.2	TLB testing revised.
2.12	Debugging support revised.
3.1	LOCK prefix restricted to certain instructions.
4.4.3.3	I/O privilege level and I/O permission bitmap added.
Figures 4-15a, 4-15b	I/O permission bitmap added.
4.6.4	Protection and I/O permission bitmap revised.
4.6.6	Entering and leaving virtual 8086 mode through task switches, trap and interrupt gates, and IRET explained.
5.6	Self-test signature stored in EAX.
5.8	Coprocessor interface description added.
5.8.1	Software testing for coprocessor presence added.
Table 6-3	PGA package thermal characteristics added.
7.	Designing for ICE-386 revised.
Figures 7-8, 7-9, 7-10	ICE-386 clearance requirements added.
6.2.3.4	Encoding of 32-bit address mode with no "sib" byte corrected.

The sections significantly revised since version -002 are:

Table 2-5	Interrupt vector assignments updated.
Figure 4-15a	Bit_map_offset must be less than or equal to DFFFH.
Figure 5-28	386 DX outputs remain in their reset state during self-test.
5.7	Component and revision identifier history updated.
9.4	20 MHz D.C. specifications added.
9.5	16 MHz A.C. specifications updated. 20 MHz A.C. specifications added.
Table 6-1	Clock counts updated.

The sections significantly revised since version -003 are:

Table 2-6b	Interrupt priorities 2 and 3 interchanged.
2.9.8	Double page faults do not raise double fault exception.
Figure 4-5	Maximum-sized segments must have segments Base _{11..0} = 0.
5.4.3.4	BS16# timing corrected.
Figures 5-16, 5-17, 5-19, 5-22	BS16# timing corrected. BS16# must not be asserted once NA# has been sampled asserted in the current bus cycle.
9.5	16 MHz and 20 MHz A.C. specifications revised. All timing parameters are now guaranteed at 1.5V test levels. The timing parameters have been adjusted to remain compatible with previous 0.8V/2.0V specifications.

The sections significantly revised since version -004 are:

Chapter 4	25 MHz Clock data included.
Table 2-4	Segment Register Selection Rules updated.
5.4.4	Interrupt Acknowledge Cycles discussion corrected.
Table 5-10	Additional Stepping Information added.
Table 9-3	I _{CC} values updated.
9.5.2	Table for 25 MHz A.C. Characteristics added. A.C. Characteristics tables reordered.
Figure 9-5	Output Valid Delay Timing Figure reconfigured. Partial data now provided in additional Figures 9-5a and 9-5b.
Table 6-1	Clock counts updated and formats corrected.

The sections significantly revised since version -005 are:

Table of Contents	Simplified.
Chapter 1	Pin Assignment.
2.3.6	Control Register 0.
Table 2-4	Segment override prefixes possible.
Figure 4-6	Note added.
Figure 4-7	Note added.
5.2.3	Data bus state at end of cycle.
5.2.8.4	Coprocessor error.
5.5.3	Bus activity during and following reset.
Figure 5-28	ERROR#.
Chapter 6	Moved forward in datasheet.
Chapter 7	Moved forward in datasheet.
Chapter 8	Upgraded to chapter.
Table 9-3	25 MHz I _{CC} Typ. value corrected.
Table 9-3	33 MHz D.C. Specifications added.
Table 9-4	33 MHz A.C. Specifications added.
Figure 9-5	t _{8a} and t _{10a} added.
Figure 9-5c	Added.
9.5.6	Added derating for C _L = 75 pF.
9.5.7	Added derating for C _L = 50 pF.
Figure 9.6	t _{8a} and t _{10a} added.

The sections significantly revised since version -006 are:

2.3.4	Alignment of maximum sized segments.
2.9.8	Double page faults do not raise double fault exception.
5.5.3	ERROR# and BUSY# sampling after RESET.
Figure 5-21	BS16# timing altered.
Figure 5-26	READY# timing altered.
Figure 5-28	ERROR# timing corrected.
6.2.3.1	Corrected Encoding of Register Field Chart.
Chapter 7	Updated ICE-386 DX information.
9.5.2	Remove preliminary stamp on 25 MHz A.C. Specifications.
9.5.2	Remove preliminary stamp on 33 MHz A.C. Specifications.

387™ DX MATH COPROCESSOR

- High Performance 80-Bit Internal Architecture
- Implements ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic
- Six to Eleven Times 8087/80287 Performance
- Expands 386™ DX CPU Data Types to Include 32-, 64-, 80-Bit Floating Point, 32-, 64-Bit Integers and 18-Digit BCD Operands
- Directly Extends 386™ DX CPU Instruction Set to include Trigonometric, Logarithmic, Exponential and Arithmetic Instructions for All Data Types
- Upward Object-Code Compatible from 8087 and 80287
- Full-Range Transcendental Operations for SINE, COSINE, TANGENT, ARCTANGENT and LOGARITHM
- Built-In Exception Handling
- Operates Independently of Real, Protected and Virtual-8086 Modes of the 386™ DX Microprocessor
- Eight 80-Bit Numeric Registers, Usable as Individually Addressable General Registers or as a Register Stack
- Available in 68-Pin PGA Package
(See Packaging Spec: Order #231369)

The Intel 387™ DX Math CoProcessor is an extension to the Intel 386™ microprocessor architecture. The combination of the 387 DX with the 386™ DX Microprocessor dramatically increases the processing speed of computer application software which utilize mathematical operations. This makes an ideal computer workstation platform for applications such as financial modeling and spreadsheets, CAD/CAM, or graphics.

The 387 DX Math CoProcessor adds over seventy mnemonics to the 386 DX Microprocessor instruction set. Specific 387 DX math operations include logarithmic, arithmetic, exponential, and trigonometric functions. The 387 DX supports integer, extended integer, floating point and BCD data formats, and fully conforms to the ANSI/IEEE floating point standard.

The 387 DX Math CoProcessor is object code compatible with the 80387SX, and upward object code compatible from the 80287 and 8087 math coprocessors. Object code for 386 DX/387 DX is also compatible with the Intel 486™ microprocessor. The 387 DX is manufactured on 1 micron, CHMOS IV technology and packaged in a 68-pin PGA package.

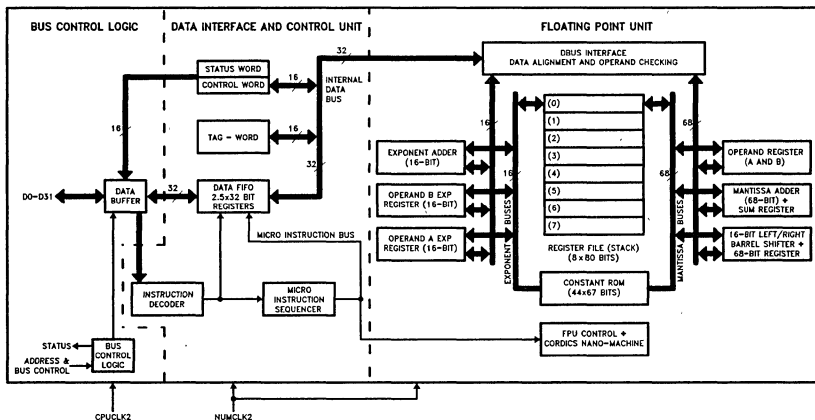


Figure 0.1. 387™ DX Math Coprocessor Block Diagram

240448-1

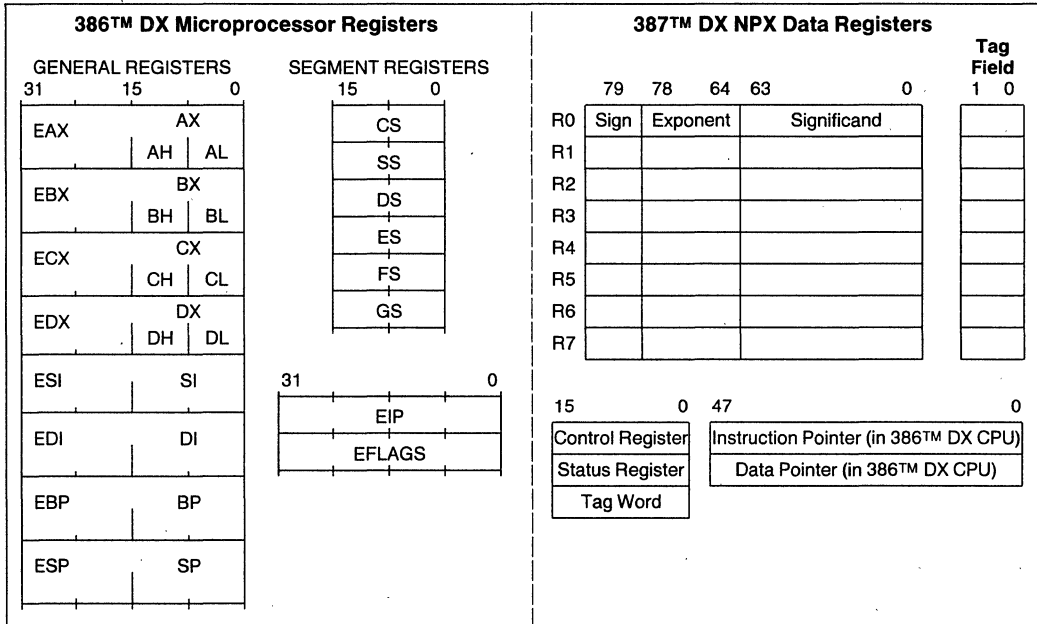


Figure 1.1. 386™ DX Microprocessor and 387™ DX Math Coprocessor Register Set

1.0 FUNCTIONAL DESCRIPTION

The 387™ DX Math Coprocessor provides arithmetic instructions for a variety of numeric data types in 386™ DX Microprocessor and 387 DX Math Coprocessor systems. It also executes numerous built-in transcendental functions (e.g. tangent, sine, cosine, and log functions). The 387 DX Math Coprocessor effectively extends the register and instruction set of a 386 DX Microprocessor system for existing data types and adds several new data types as well. Figure 1.1 shows the model of registers visible to 386 DX Microprocessor and 387 DX Math Coprocessor programs. Essentially, the 387 DX Math Coprocessor can be treated as an additional resource or an extension to the 386 DX Microprocessor. The 386 DX Microprocessor together with a 387 DX Math Coprocessor can be used as a single unified system, the 386 DX Microprocessor and 387 DX Math Coprocessor.

The 387 DX Math Coprocessor works the same whether the 386 DX Microprocessor is executing in real-address mode, protected mode, or virtual-8086 mode. All memory access is handled by the 386 DX Microprocessor; the 387 DX Math Coprocessor merely operates on instructions and values passed to it by the 386 DX Microprocessor. Therefore, the 387 DX Math Coprocessor is not sensitive to the processing mode of the 386 DX Microprocessor.

In real-address mode and virtual-8086 mode, the 386 DX Microprocessor and 387 DX Math Coprocessor are completely upward compatible with software for 8086/8087, 80286/80287 real-address mode, and 386 DX Microprocessor and 80287 Coprocessor real-address mode systems.

In protected mode, the 386 DX Microprocessor and 387 DX Math Coprocessor are completely upward compatible with software for 80286/80287 protected mode, and 386 DX Microprocessor and 80287 Coprocessor protected mode systems.

The only differences of operation that may appear when 8086/8087 programs are ported to a protected-mode 386 DX Microprocessor and 387 DX Math Coprocessor system (*not* using virtual-8086 mode), is in the format of operands for the administrative instructions FLDENV, FSTENV, FRSTOR and FSAVE. These instructions are normally used only by exception handlers and operating systems, not by applications programs.

The 387 DX Math Coprocessor contains three functional units that can operate in parallel to increase system performance. The 386 DX Microprocessor can be transferring commands and data to the NPX *bus control logic* for the next instruction while the NPX *floating-point unit* is performing the current numeric instruction.

2.0 PROGRAMMING INTERFACE

The NPX adds to the 386 DX Microprocessor system additional data types, registers, instructions, and interrupts specifically designed to facilitate high-speed numerics processing. To use the NPX requires no special programming tools, because all new instructions and data types are directly supported by the 386 DX CPU assembler and compilers for high-level languages. All 8086/8088 development tools that support the 8087 can also be used to develop software for the 386 DX Microprocessor and 387 DX Math Coprocessor in real-address mode or virtual-8086 mode. All 80286 development tools that support the 80287 can also be used to develop software for the 386 DX Microprocessor and 387 DX Math Coprocessor.

All communication between the 386 DX Microprocessor and the NPX is transparent to applications software. The CPU automatically controls the NPX whenever a numerics instruction is executed. All physical memory and virtual memory of the CPU are available for storage of the instructions and operands of programs that use the NPX. All memory addressing modes, including use of displacement, base register, index register, and scaling, are available for addressing numerics operands.

Section 6 at the end of this data sheet lists by class the instructions that the NPX adds to the instruction set of the 386 DX Microprocessor system.

2.1 Data Types

Table 2.1 lists the seven data types that the 387 DX NPX supports and presents the format for each type. Operands are stored in memory with the least significant digit at the lowest memory address. Programs retrieve these values by generating the lowest address. For maximum system performance, all operands should start at physical-memory addresses evenly divisible by four (doubleword boundaries); operands may begin at any other addresses, but will require extra memory cycles to access the entire operand.

Internally, the 387 DX NPX holds all numbers in the extended-precision real format. Instructions that load operands from memory automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating-point numbers, or 18-digit packed BCD numbers into extended-precision real format. Instructions that store operands in memory perform the inverse type conversion.

2.2 Numeric Operands

A typical NPX instruction accepts one or two operands and produces a single result. In two-operand instructions, one operand is the contents of an NPX register, while the other may be a memory location. The operands of some instructions are predefined; for example FSQRT always takes the square root of the number in the top stack element.

Table 2.1. 387™ DX NPX Data Type Representation in Memory

Data Formats	Range	Precision	Most Significant Byte = Highest Addressed Byte																																																															
			7	0	7	0	7	0	7	0	7	0	7	0	7	0	7	0																																																
Word Integer	$\pm 10^4$	16 Bits	<div style="border: 1px solid black; padding: 2px; display: inline-block;"> (TWO'S COMPLEMENT) 15 0 </div>																																																															
Short Integer	$\pm 10^9$	32 Bits	<div style="border: 1px solid black; padding: 2px; display: inline-block;"> (TWO'S COMPLEMENT) 31 0 </div>																																																															
Long Integer	$\pm 10^{18}$	64 Bits	<div style="border: 1px solid black; padding: 2px; display: inline-block;"> (TWO'S COMPLEMENT) 63 0 </div>																																																															
Packed BCD	$\pm 10^{\pm 18}$	18 Digits	S	X	MAGNITUDE																																																													
			79	72	d_{17}	d_{16}	d_{15}	d_{14}	d_{13}	d_{12}	d_{11}	d_{10}	d_9	d_8	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	0																																											
Single Precision	$\pm 10^{\pm 38}$	24 Bits	S	BIASED EXPONENT				SIGNIFICAND																																																										
			31	23				0																																																										
			<div style="text-align: center;"> $\leftarrow I_{\Delta}$ </div>																																																															
Double Precision	$\pm 10^{\pm 308}$	53 Bits	S	BIASED EXPONENT								SIGNIFICAND																																																						
			63	52								0																																																						
			<div style="text-align: center;"> $\leftarrow I_{\Delta}$ </div>																																																															
Extended Precision	$\pm 10^{\pm 4932}$	64 Bits	S	BIASED EXPONENT										1	SIGNIFICAND																																																			
			79	64										63	0																																																			
			<div style="text-align: center;"> $\leftarrow I_{\Delta}$ </div>																																																															

240448-2

NOTES:

- (1) S = Sign bit (0 = positive, 1 = negative)
- (2) d_n = Decimal digit (two per byte)
- (3) X = Bits have no significance; 387™ DX NPX ignores when loading, zeros when storing
- (4) Δ = Position of implicit binary point
- (5) I = Integer bit of significand; stored in temporary real, implicit in single and double precision
- (6) Exponent Bias (normalized values):
 Single: 127 (7FH)
 Double: 1023 (3FFH)
 Extended Real: 16383 (3FFFH)
- (7) Packed BCD: $(-1)^S (D_{17}...D_0)$
- (8) Real: $(-1)^S (2^E \text{-BIAS}) (F_0 F_1...)$

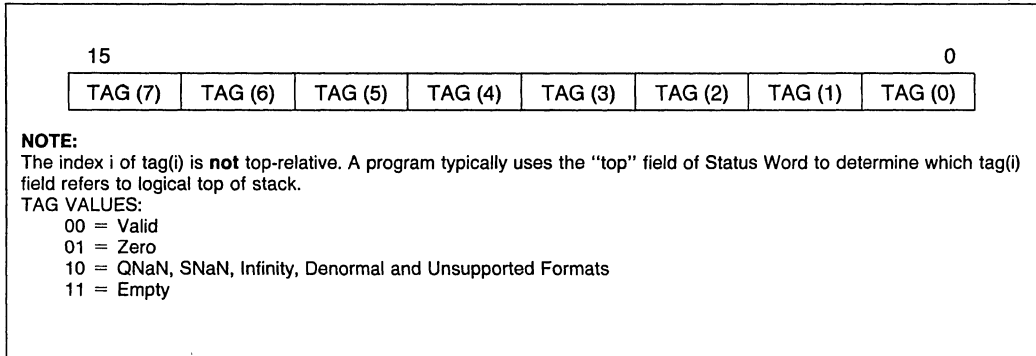


Figure 2.1. 387™ DX NPX Tag Word

2.3 Register Set

Figure 1.1 shows the 387 DX NPX register set. When an NPX is present in a system, programmers may use these registers in addition to the registers normally available on the 386 DX CPU.

2.3.1 DATA REGISTERS

387 DX NPX computations use the NPX's data registers. These eight 80-bit registers provide the equivalent capacity of twenty 32-bit registers. Each of the eight data registers in the NPX is 80 bits wide and is divided into "fields" corresponding to the NPXs extended-precision real data type.

The 387 DX NPX register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers. The TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by one and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments

TOP by one. Like the 386 DX Microprocessor stacks in memory, the NPX register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to use. This explicit register addressing is also relative to TOP.

2.3.2 TAG WORD

The tag word marks the content of each numeric data register, as Figure 2.1 shows. Each two-bit tag represents one of the eight numeric registers. The principal function of the tag word is to optimize the NPXs performance and stack handling by making it possible to distinguish between empty and nonempty register locations. It also enables exception handlers to check the contents of a stack location without the need to perform complex decoding of the actual data.

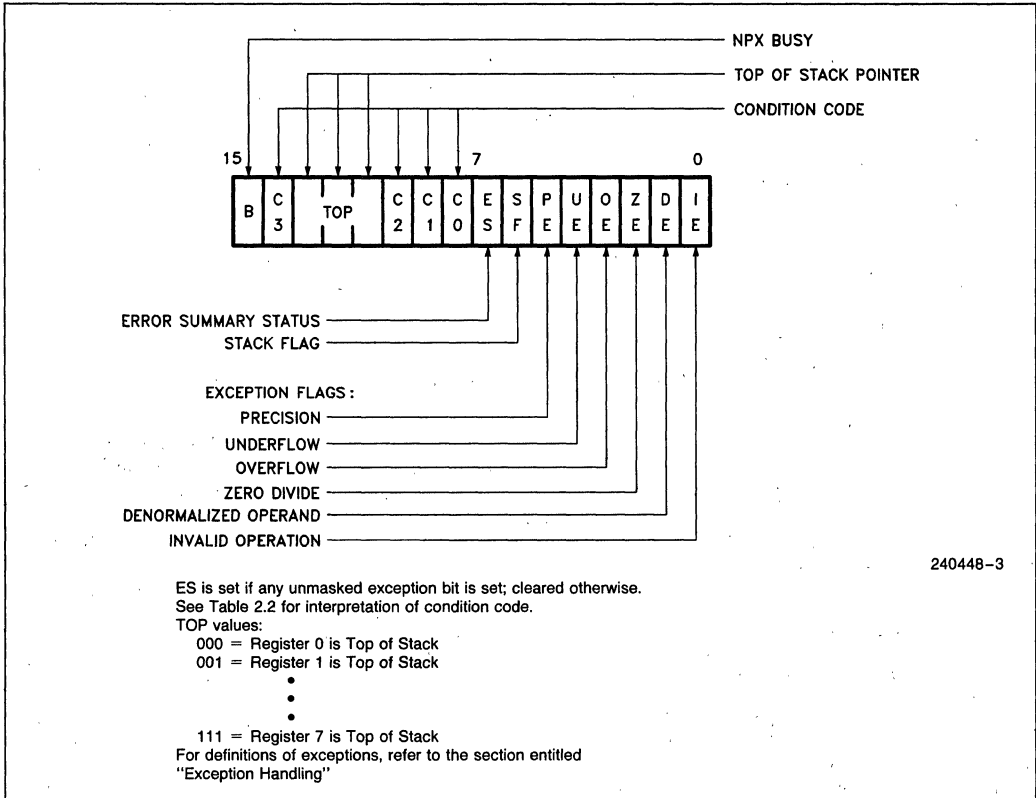


Figure 2.2. NPX Status Word

2.3.3 STATUS WORD

The 16-bit status word (in the status register) shown in Figure 2.2 reflects the overall state of the NPX. It may be read and inspected by CPU code.

Bit 15, the B-bit (busy bit) is included for 8087 compatibility only. It reflects the contents of the ES bit (bit 7 of the status word), not the status of the BUSY# output of the 387 DX NPX.

Bits 13-11 (TOP) point to the 387 DX NPX register that is the current top-of-stack.

The four numeric condition code bits (C₃-C₀) are similar to the flags in a CPU; instructions that perform arithmetic operations update these bits to reflect the outcome. The effects of these instructions on the condition code are summarized in Tables 2.2 through 2.5.

Bit 7 is the error summary (ES) status bit. This bit is set if any unmasked exception bit is set; it is clear otherwise. If this bit is set, the ERROR# signal is asserted.

Bit 6 is the stack flag (SF). This bit is used to distinguish invalid operations due to stack overflow or underflow from other kinds of invalid operations. When SF is set; bit 9 (C₁) distinguishes between stack overflow (C₁ = 1) and underflow (C₁ = 0).

Figure 2.2 shows the six exception flags in bits 5-0 of the status word. Bits 5-0 are set to indicate that the NPX has detected an exception while executing an instruction. A later section entitled "Exception Handling" explains how they are set and used.

Note that when a new value is loaded into the status word by the FLDENV or FRSTOR instruction, the value of ES (bit 7) and its reflection in the B-bit (bit 15) are not derived from the values loaded from memory but rather are dependent upon the values of the exception flags (bits 5-0) in the status word and their corresponding masks in the control word. If ES is set in such a case, the ERROR# output of the NPX is activated immediately.

Table 2.2. Condition Code Interpretation

Instruction	C0 (S)	C3 (Z)	C1 (A)	C2 (C)
FPREM, FPREM1 (see Table 2.3)	Three least significant bits of quotient Q2 Q0			Reduction 0 = complete 1 = incomplete Q1 or O/U#
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 2.4)		Zero or O/U#	Operand is not comparable (Table 2.4)
FXAM	Operand class (see Table 2.5)		Sign or O/U#	Operand class (Table 2.5)
FCHS, FABS, FXCH, FINCSTP, FDECSTP, Constant loads, FEXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/U#	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/U#	UNDEFINED
FPTAN, FSIN FCOS, FSINCOS	UNDEFINED		Roundup or O/U#, undefined if C2 = 1	Reduction 0 = complete 1 = incomplete
FLDENV, FRSTOR	Each bit loaded from memory			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FINIT, FSAVE	UNDEFINED			
O/U#	When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).			
Reduction	If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS, and FSINCOS, the reduction bit is set if the operand at the top of the stack is too large. In this case the original operand remains at the top of the stack.			
Roundup	When the PE bit of the status word is set, this bit indicates whether the last rounding in the instruction was upward.			
UNDEFINED	Do not rely on finding any specific value in these bits.			

Table 2.3. Condition Code Interpretation after FPREM and FPREM1 Instructions

Condition Code				Interpretation after FPREM and FPREM1	
C2	C3	C1	C0		
1	X	X	X	Incomplete Reduction: further iteration required for complete reduction	
0	Q1	Q0	Q2	Q MOD8	Complete Reduction: C0, C3, C1 contain three least significant bits of quotient
	0	0	0	0	
	0	1	0	1	
	1	0	0	2	
	1	1	0	3	
	0	0	1	4	
	0	1	1	5	
	1	0	1	6	
1	1	1	7		

Table 2.4. Condition Code Resulting from Comparison

Order	C3	C2	C0
TOP > Operand	0	0	0
TOP < Operand	0	0	1
TOP = Operand	1	0	0
Unordered	1	1	1

Table 2.5. Condition Code Defining Operand Class

C3	C2	C1	C0	Value at TOP
0	0	0	0	+ Unsupported
0	0	0	1	+ NaN
0	0	1	0	Unsupported
0	0	1	1	- NaN
0	1	0	0	+ Normal
0	1	0	1	+ Infinity
0	1	1	0	- Normal
0	1	1	1	- Infinity
1	0	0	0	+ 0
1	0	0	1	+ Empty
1	0	1	0	- 0
1	0	1	1	- Empty
1	1	0	0	+ Denormal
1	1	1	0	- Denormal

2.3.4 INSTRUCTION AND DATA POINTERS

Because the NPX operates in parallel with the CPU, any errors detected by the NPX may be reported after the CPU has executed the ESC instruction which caused it. To allow identification of the failing numeric instruction, the 386 DX Microprocessor and 387 DX Math Coprocessor contains two pointer registers that supply the address of the failing numeric instruction and the address of its numeric memory operand (if appropriate).

The instruction and data pointers are provided for user-written error handlers. These registers are actually located in the 386 DX CPU, but appear to be located in the NPX because they are accessed by the ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR. (In the 8086/8087 and 80286/80287, these registers are located in the NPX.) Whenever

the 386 DX CPU decodes a new ESC instruction, it saves the address of the instruction (including any prefixes that may be present), the address of the operand (if present), and the opcode.

The instruction and data pointers appear in one of four formats depending on the operating mode of the 386 DX Microprocessor (protected mode or real-address mode) and depending on the operand-size attribute in effect (32-bit operand or 16-bit operand). When the 386 DX Microprocessor is in virtual-8086 mode, the real-address mode formats are used. (See Figures 2.3 through 2.6.) The ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR are used to transfer these values between the 386 DX Microprocessor registers and memory. Note that the value of the data pointer is *undefined* if the prior ESC instruction did not have a memory operand.

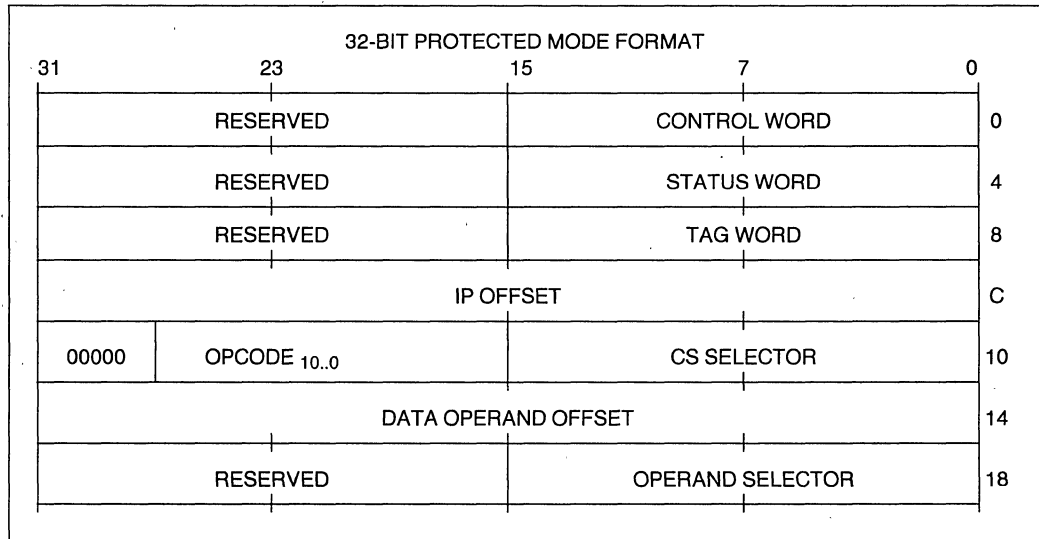


Figure 2.3. Protected Mode 387™ DX NPX Instruction and Data Pointer Image in Memory, 32-Bit Format

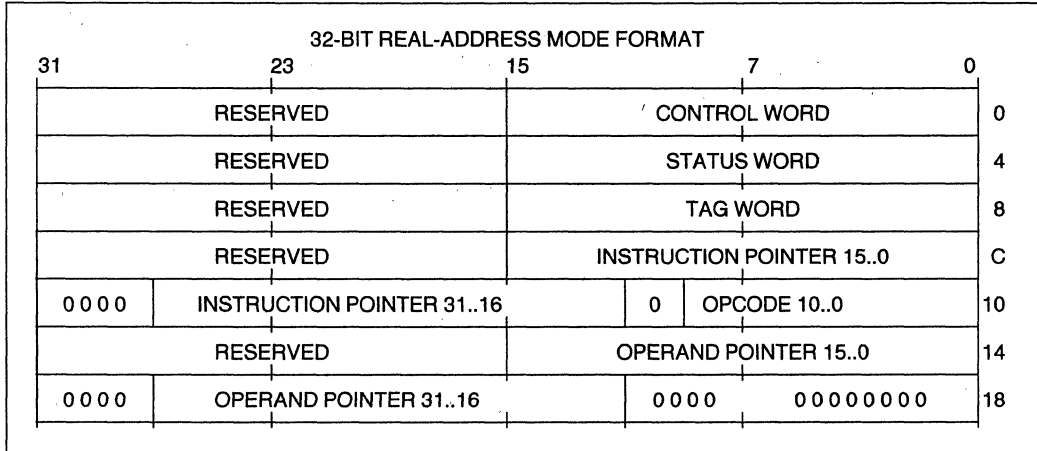


Figure 2.4. Real Mode 387™ DX NPX Instruction and Data Pointer Image in Memory, 32-Bit Format

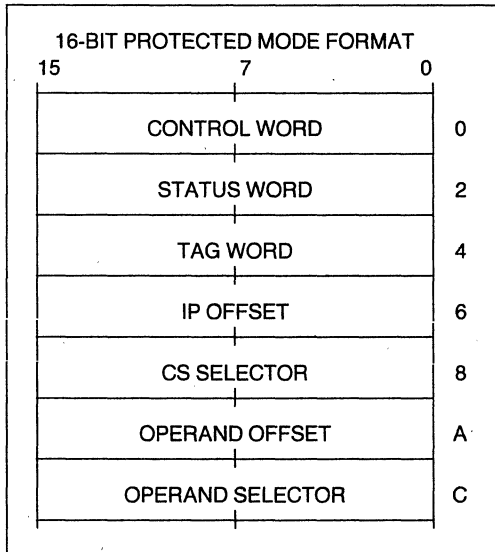


Figure 2.5. Protected Mode 387™ DX NPX Instruction and Data Pointer Image in Memory, 16-Bit Format

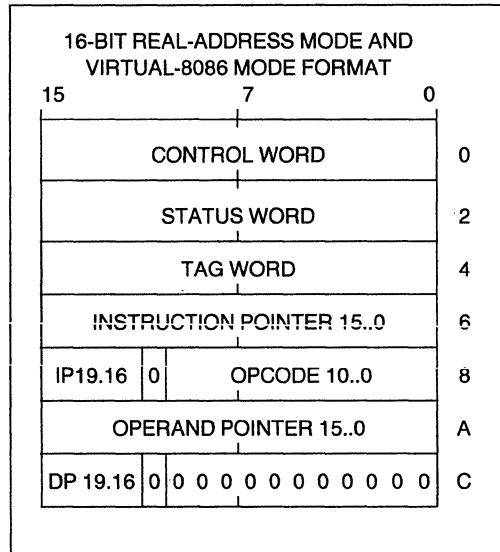


Figure 2.6. Real Mode 387™ DX NPX Instruction and Data Pointer Image in Memory, 16-Bit Format

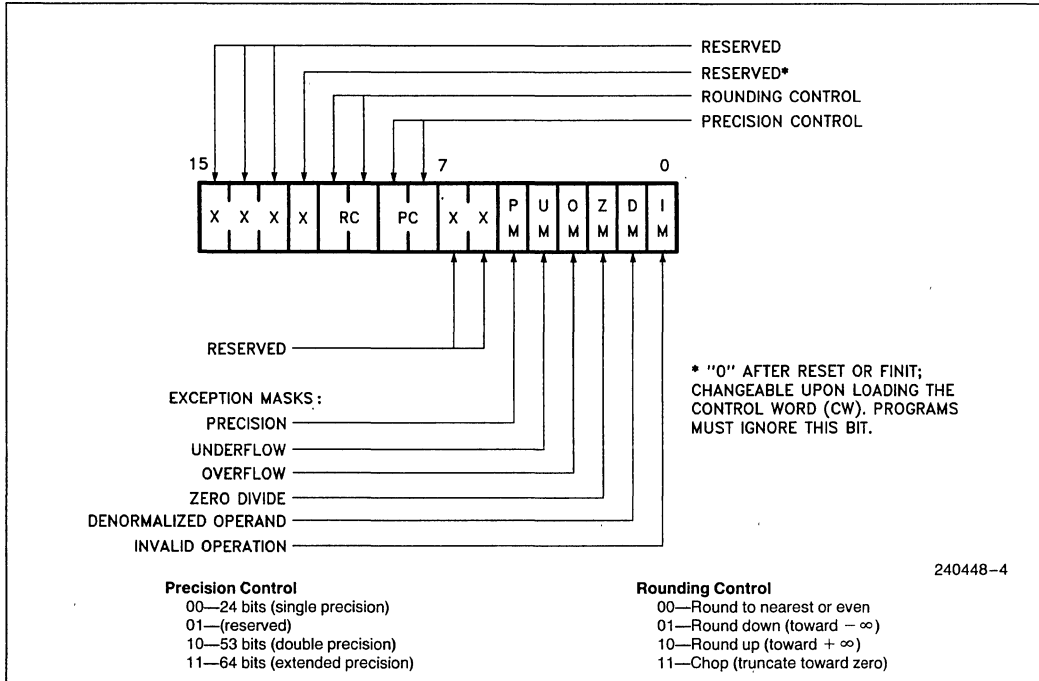


Figure 2.7. 387™ DX NPX Control Word

2.3.5 CONTROL WORD

The NPX provides several processing options that are selected by loading a control word from memory into the control register. Figure 2.7 shows the format and encoding of fields in the control word.

The low-order byte of this control word configures the NPX error and exception masking. Bits 5–0 of the control word contain individual masks for each of the six exceptions that the NPX recognizes.

The high-order byte of the control word configures the NPX operating mode, including precision and rounding.

- Bit 12 no longer defines infinity control and is a reserved bit. Only affine closure is supported for infinity arithmetic. The bit is initialized to zero after RESET or FINIT and is changeable upon loading the CW. Programs must ignore this bit.
- The rounding control (RC) bits (bits 11–10) provide for directed rounding and true chop, as well as the unbiased round to nearest even mode specified in the IEEE standard. Rounding control

affects only those instructions that perform rounding at the end of the operation (and thus can generate a precision exception); namely, FST, FSTP, FIST, all arithmetic instructions (except FPREM, FPREM1, FXTRACT, FABS, and FCHS), and all transcendental instructions.

- The precision control (PC) bits (bits 9–8) can be used to set the NPX internal operating precision of the significand at less than the default of 64 bits (extended precision). This can be useful in providing compatibility with early generation arithmetic processors of smaller precision. PC affects only the instructions ADD, SUB, DIV, MUL, and SQRT. For all other instructions, either the precision is determined by the opcode or extended precision is used.

2.4 Interrupt Description

Several interrupts of the 386 DX CPU are used to report exceptional conditions while executing numeric programs in either real or protected mode. Table 2.6 shows these interrupts and their causes.

Table 2.6. 386™ DX Microprocessor Interrupt Vectors Reserved for NPX

Interrupt Number	Cause of Interrupt
7	An ESC instruction was encountered when EM or TS of the 386™ DX CPU control register zero (CR0) was set. EM = 1 indicates that software emulation of the instruction is required. When TS is set, either an ESC or WAIT instruction causes interrupt 7. This indicates that the current NPX context may not belong to the current task.
9	An operand of a coprocessor instruction wrapped around an addressing limit (0FFFFH for small segments, 0FFFFFFFH for big segments, zero for expand-down segments) and spanned inaccessible addresses ^a . The failing numerics instruction is not restartable. The address of the failing numerics instruction and data operand may be lost; an FSTENV does not return reliable addresses. As with the 80286/80287, the segment overrun exception should be handled by executing an FNINIT instruction (i.e. an FINIT without a preceding WAIT). The return address on the stack does not necessarily point to the failing instruction nor to the following instruction. The interrupt can be avoided by never allowing numeric data to start within 108 bytes of the end of a segment.
13	The first word or doubleword of a numeric operand is not entirely within the limit of its segment. The return address pushed onto the stack of the exception handler points at the ESC instruction that caused the exception, including any prefixes. The 387™ DX NPX has not executed this instruction; the instruction pointer and data pointer register refer to a previous, correctly executed instruction.
16	The previous numerics instruction caused an unmasked exception. The address of the faulty instruction and the address of its operand are stored in the instruction pointer and data pointer registers. Only ESC and WAIT instructions can cause this interrupt. The 386™ DX CPU return address pushed onto the stack of the exception handler points to a WAIT or ESC instruction (including prefixes). This instruction can be restarted after clearing the exception condition in the NPX. FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE cannot cause this interrupt.

a. An operand may wrap around an addressing limit when the segment limit is near an addressing limit and the operand is near the largest valid address in the segment. Because of the wrap-around, the beginning and ending addresses of such an operand will be at opposite ends of the segment. There are two ways that such an operand may also span inaccessible addresses: 1) if the segment limit is not equal to the addressing limit (e.g. addressing limit is FFFFH and segment limit is FFFDH) the operand will span addresses that are not within the segment (e.g. an 8-byte operand that starts at valid offset FFFC will span addresses FFFC-FFFF and 0000-0003; however addresses FFFE and FFFF are not valid, because they exceed the limit); 2) if the operand begins and ends in present and accessible pages but intermediate bytes of the operand fall in a not-present page or a page to which the procedure does not have access rights.

2.5 Exception Handling

The 387 DX NPX detects six different exception conditions that can occur during instruction execution. Table 2.7 lists the exception conditions in order of precedence, showing for each the cause and the default action taken by the NPX if the exception is masked by its corresponding mask bit in the control word.

Any exception that is not masked by the control word sets the corresponding exception flag of the status word, sets the ES bit of the status word, and asserts the ERROR# signal. When the CPU attempts to execute another ESC instruction or WAIT, exception 7 occurs. The exception condition must be resolved via an interrupt service routine. The 386 DX Microprocessor and 387 DX Math Coprocessor save the address of the floating-point instruction that caused the exception and the address of any memory operand required by that instruction.

2.6 Initialization

387 DX NPX initialization software must execute an FNINIT instruction (i.e. an FINIT without a preceding WAIT) to clear ERROR#. After a hardware RESET, the ERROR# output is asserted to indicate that a 387 DX NPX is present. To accomplish this, the IE and ES bits of the status word are set, and the IM bit in the control word is reset. After FNINIT, the status word and the control word have the same values as in an 80287 after RESET.

2.7 8087 and 80287 Compatibility

This section summarizes the differences between the 387 DX NPX and the 80287. Any migration from the 8087 directly to the 387 DX NPX must also take into account the differences between the 8087 and the 80287 as listed in Appendix A.

Many changes have been designed into the 387 DX NPX to directly support the IEEE standard in hardware. These changes result in increased performance by eliminating the need for software that supports the standard.

2.7.1 GENERAL DIFFERENCES

The 387 DX NPX supports only affine closure for infinity arithmetic, not projective closure. Bit 12 of the Control Word (CW) no longer defines infinity control. It is a reserved bit; but it is initialized to zero after RESET or FINIT and is changeable upon loading the CW. Programs must ignore this bit.

Operands for FSCALE and FPATAN are no longer restricted in range (except for $\pm\infty$); F2XM1 and FPTAN accept a wider range of operands.

The results of transcendental operations may be slightly different from those computed by 80287.

In the case of FPTAN, the 387 DX NPX supplies a true tangent result in ST(1), and (always) a floating point 1 in ST.

Rounding control is in effect for FLD *constant*.

Software cannot change entries of the tag word to values (other than empty) that do not reflect the actual register contents.

After reset, FINIT, and incomplete FPREM, the 387 DX NPX resets to zero the condition code bits C₃-C₀ of the status word.

In conformance with the IEEE standard, the 387 DX NPX does not support the special data formats: pseudozero, pseudo-NaN, pseudoinfinity, and unnormal.

Table 2.7. Exceptions

Exception	Cause	Default Action (if exception is masked)
Invalid Operation	Operation on a signaling NaN, unsupported format, indeterminate form ($0*\infty$, $0/0$, $(+\infty) + (-\infty)$, etc.), or stack overflow/underflow (SF is also set).	Result is a quiet NaN, integer indefinite, or BCD indefinite
Denormalized Operand	At least one of the operands is denormalized, i.e. it has the smallest exponent but a nonzero significand.	Normal processing continues
Zero Divisor	The divisor is zero while the dividend is a noninfinite, nonzero number.	Result is ∞
Overflow	The result is too large in magnitude to fit in the specified format.	Result is largest finite value or ∞
Underflow	The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes loss of accuracy.	Result is denormalized or zero
Inexact Result (Precision)	The true result is not exactly representable in the specified format (e.g. 1/3); the result is rounded according to the rounding mode.	Normal processing continues

2.7.2 EXCEPTIONS

A number of differences exist due to changes in the IEEE standard and to functional improvements to the architecture of the 387 DX NPX:

1. When the overflow or underflow exception is masked, the 387 DX NPX differs from the 80287 in rounding when overflow or underflow occurs. The 387 DX NPX produces results that are consistent with the rounding mode.
2. When the underflow exception is masked, the 387 DX NPX sets its underflow flag only if there is also a loss of accuracy during denormalization.
3. Fewer invalid-operation exceptions due to denormal operands, because the instructions FSQRT, FDIV, FPREM, and conversions to BCD or to integer normalize denormal operands before proceeding.
4. The FSQRT, FBSTP, and FPREM instructions may cause underflow, because they support denormal operands.
5. The denormal exception can occur during the transcendental instructions and the FTRACT instruction.
6. The denormal exception no longer takes precedence over all other exceptions.
7. When the denormal exception is masked, the 387 DX NPX automatically normalizes denormal operands. The 8087/80287 performs unnormal arithmetic, which might produce an unnormal result.
8. When the operand is zero, the FTRACT instruction reports a zero-divide exception and leaves $-\infty$ in ST(1).
9. The status word has a new bit (SF) that signals when invalid-operation exceptions are due to stack underflow or overflow.
10. FLD *extended precision* no longer reports denormal exceptions, because the instruction is not numeric.
11. FLD *single/double precision* when the operand is denormal converts the number to extended precision and signals the denormalized operand exception. When loading a signaling NaN, FLD *single/double precision* signals an invalid-operation exception.
12. The 387 DX NPX only generates quiet NaNs (as on the 80287); however, the 387 DX NPX distinguishes between quiet NaNs and signaling NaNs. Signaling NaNs trigger exceptions when they are used as operands; quiet NaNs do not (except for FCOM, FIST, and FBSTP which also raise IE for quiet NaNs).
13. When stack overflow occurs during FPTAN and overflow is masked, both ST(0) and ST(1) contain quiet NaNs. The 80287/8087 leaves the original operand in ST(1) intact.
14. When the scaling factor is $\pm\infty$, the FSCALE (ST(0), ST(1)) instruction behaves as follows (ST(0) and ST(1) contain the scaled and scaling operands respectively):
 - FSCALE(0, ∞) generates the invalid operation exception.
 - FSCALE(finite, $-\infty$) generates zero with the same sign as the scaled operand.
 - FSCALE(finite, $+\infty$) generates ∞ with the same sign as the scaled operand.

The 8087/80287 returns zero in the first case and raises the invalid-operation exception in the other cases.
15. The 387 DX NPX returns signed infinity/zero as the unmasked response to massive overflow/underflow. The 8087 and 80287 support a limited range for the scaling factor; within this range either massive overflow/underflow do not occur or undefined results are produced.

3.0 HARDWARE INTERFACE

In the following description of hardware interface, the # symbol at the end of a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no # is present after the signal name, the signal is asserted when at the high voltage level.

3.1 Signal Description

In the following signal descriptions, the 387 DX Math Coprocessor pins are grouped by function as follows:

1. Execution control—CPUCLK2, NUMCLK2, CKM, RESETIN
2. NPX handshake—PEREQ, BUSY#, ERROR#
3. Bus interface pins—D31–D0, W/R#, ADS#, READY#, READYO#
4. Chip/Port Select—STEN, NPS1#, NPS2, CMD0#
5. Power supplies—V_{CC}, V_{SS}

Table 3.1 lists every pin by its identifier, gives a brief description of its function, and lists some of its characteristics. All output signals are tristate; they leave floating state only when STEN is active. The output buffers of the bidirectional data pins D31–D0 are also tristate; they leave floating state only in read cycles when the NPX is selected (i.e. when STEN, NPS1#, and NPS2 are all active).

Figure 3.1 and Table 3.2 together show the location of every pin in the pin grid array.

Table 3.1. 387™ DX NPX Pin Summary

Pin Name	Function	Active State	Input/Output	Referenced To
CPUCLK2 NUMCLK2 CKM RESETIN	386™ DX CPU CLock 2 387™ DX NPX CLock 2 387™ DX NPX CLock Mode System reset	High	I I I I	CPUCLK2
PEREQ BUSY# ERROR#	Processor Extension REQuest Busy status Error status	High Low Low	O O O	CPUCLK2/STEN CPUCLK2/STEN NUMCLK2/STEN
D31–D0 W/R# ADS# READY# READYO#	Data pins Write/Read bus cycle Address Strobe Bus ready input Ready output	High Hi/Lo Low Low Low	I/O I I I O	CPUCLK2 CPUCLK2 CPUCLK2 CPUCLK2 CPUCLK2/STEN
STEN NPS1# NPS2 CMD0#	STatus ENable NPX select #1 NPX select #2 CoMmanD	High Low High Low	I I I I	CPUCLK2 CPUCLK2 CPUCLK2 CPUCLK2
V _{CC} V _{SS}			I I	

NOTE:

STEN is referenced to only when getting the output pins into or out of tristate mode.

Table 3.2. 387™ DX NPX Pin Cross-Reference

ADS#	—	K7	D18	—	A8	STEN	—	L4
BUSY#	—	K2	D19	—	B9	W/R#	—	K4
CKM	—	J11	D20	—	B10			
CPUCLK24	—	K10	D21	—	A10	V _{CC}	—	A6, A9, B4, E1, F1, F10, J2, K3, K5, L7, L9
CMD0#	—	L8	D22	—	B11			
D0	—	H2	D23	—	C10			
D1	—	H1	D24	—	D10			
D2	—	G2	D25	—	D11			
D3	—	G1	D26	—	E10	V _{SS}	—	B2, B7, C11, E2, F2, F11, J1, J10, L5
D4	—	D2	D27	—	E11			
D5	—	D1	D28	—	G10			
D6	—	C2	D29	—	G11			
D7	—	C1	D30	—	H10	NO CONNECT	—	K9
D8	—	B1	D31	—	H11			
D9	—	A2	ERROR#	—	L2			
D10	—	B3	NPS1#	—	L6			
D11	—	A3	NPS2	—	K6			
D12	—	A4	NUMCLK2	—	K11			
D13	—	B5	PEREQ	—	K1			
D14	—	A5	READY#	—	K8			
D15	—	B6	READYO#	—	L3			
D16	—	A7	RESETIN	—	L10			
D17	—	B8						

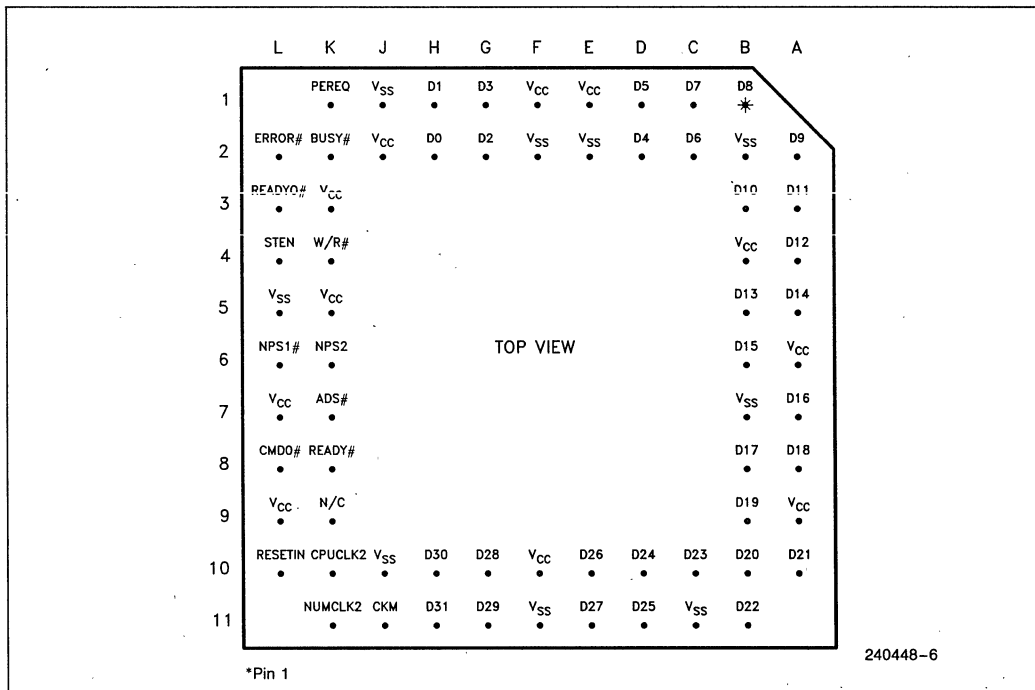
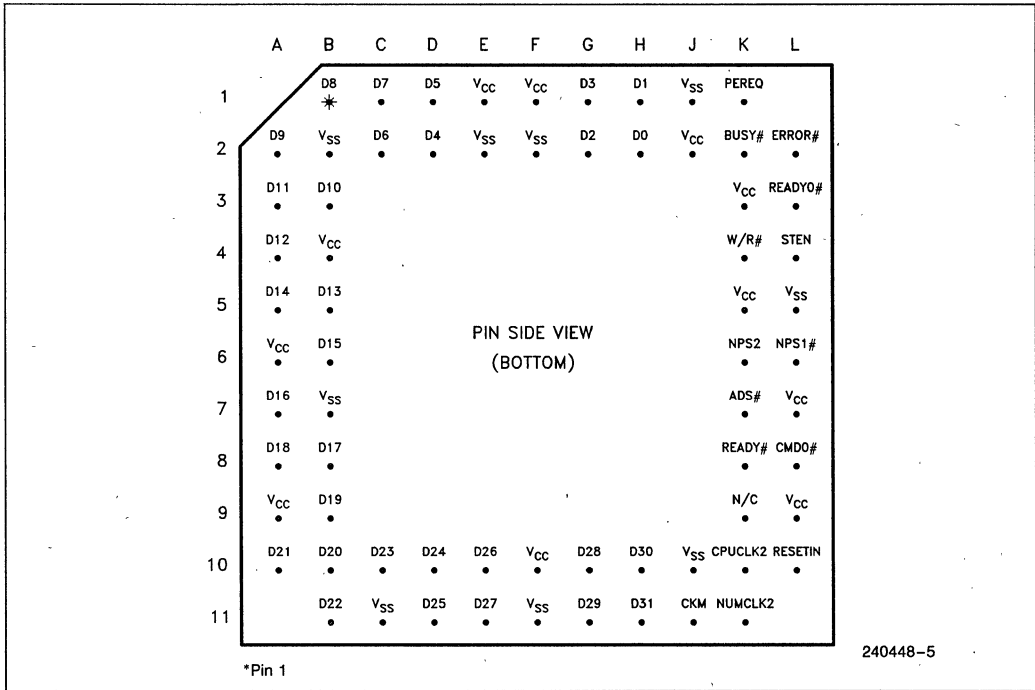


Figure 3.1. 387™ DX NPX Pin Configuration

3.1.1 386™ DX CPU CLOCK 2 (CPUCLK2)

This input uses the 386 DX CPU CLK2 signal to time the bus control logic. Several other NPX signals are referenced to the rising edge of this signal. When CKM = 1 (synchronous mode) this pin also clocks the data interface and control unit and the floating-point unit of the NPX. This pin requires MOS-level input. The signal on this pin is divided by two to produce the internal clock signal CLK.

3.1.2 387™ DX NPX CLOCK 2 (NUMCLK2)

When CKM = 0 (asynchronous mode) this pin provides the clock for the data interface and control unit and the floating-point unit of the NPX. In this case, the ratio of the frequency of NUMCLK2 to the fre-

quency of CPUCLK2 must lie within the range 10:16 to 14:10. When CKM = 1 (synchronous mode) this pin is ignored; CPUCLK2 is used instead for the data interface and control unit and the floating-point unit. This pin requires TTL-level input.

3.1.3 387™ DX NPX CLOCKING MODE (CKM)

This pin is a strapping option. When it is strapped to V_{CC}, the NPX operates in synchronous mode; when strapped to V_{SS}, the NPX operates in asynchronous mode. These modes relate to clocking of the data interface and control unit and the floating-point unit only; the bus control logic always operates synchronously with respect to the 386 DX Microprocessor.

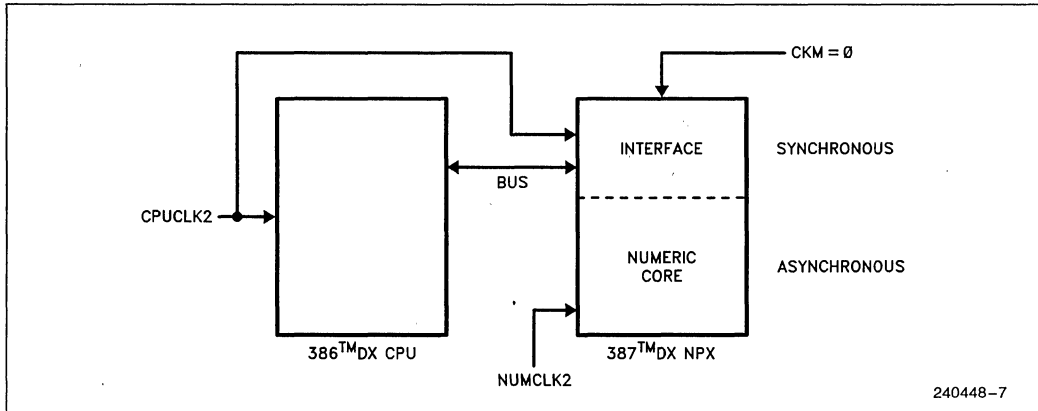


Figure 3.2. Asynchronous Operation

3.1.4 SYSTEM RESET (RESETIN)

A LOW to HIGH transition on this pin causes the NPX to terminate its present activity and to enter a dormant state. RESETIN must remain HIGH for at least 40 NUMCLK2 periods. The HIGH to LOW transitions of RESETIN must be synchronous with CPUCLK2, so that the phase of the internal clock of the bus control logic (which is the CPUCLK2 divided by 2) is the same as the phase of the internal clock of the 386 DX CPU. After RESETIN goes LOW, at least 50 NUMCLK2 periods must pass before the first NPX instruction is written into the 387 DX NPX. This pin should be connected to the 386 DX CPU RESET pin. Table 3.3 shows the status of other pins after a reset.

Table 3.3. Output Pin Status During Reset

Pin Value	Pin Name
HIGH	READYO#, BUSY#
LOW	PEREQ, ERROR#
Tri-State OFF	D31-D0

3.1.5 PROCESSOR EXTENSION REQUEST (PEREQ)

When active, this pin signals to the 386 DX CPU that the NPX is ready for data transfer to/from its data FIFO. When all data is written to or read from the data FIFO, PEREQ is deactivated. This signal always goes inactive before BUSY# goes inactive. This signal is referenced to CPUCLK2. It should be connected to the 386 DX CPU PEREQ input. Refer to Figure 3.8 for the timing relationships between this and the BUSY# and ERROR# pins.

3.1.6 BUSY STATUS (BUSY#)

When active, this pin signals to the 386 DX CPU that the NPX is currently executing an instruction. This signal is referenced to CPUCLK2. It should be connected to the 386 DX CPU BUSY# pin. Refer to Figure 3.8 for the timing relationships between this and the PEREQ and ERROR# pins.

3.1.7 ERROR STATUS (ERROR#)

This pin reflects the ES bits of the status register. When active, it indicates that an unmasked exception has occurred (except that, immediately after a reset, it indicates to the 386 DX Microprocessor that a 387 DX NPX is present in the system). This signal can be changed to inactive state only by the following instructions (without a preceding WAIT): FNINIT, FNCLEX, FNSTENV, and FNSAVE. This signal is referenced to NUMCLK2. It should be connected to the 386 DX CPU ERROR# pin. Refer to Figure 3.8 for the timing relationships between this and the PEREQ and BUSY# pins.

3.1.8 DATA PINS (D31-D0)

These bidirectional pins are used to transfer data and opcodes between the 386 DX CPU and 387 DX NPX. They are normally connected directly to the corresponding 386 DX CPU data pins. HIGH state indicates a value of one. D0 is the least significant data bit. Timings are referenced to CPUCLK2.

3.1.9 WRITE/READ BUS CYCLE (W/R#)

This signal indicates to the NPX whether the 386 DX CPU bus cycle in progress is a read or a write cycle. This pin should be connected directly to the 386 DX CPU W/R# pin. HIGH indicates a write cycle; LOW, a read cycle. This input is ignored if any of the signals STEN, NPS1#, or NPS2 is inactive. Setup and hold times are referenced to CPUCLK2.

3.1.10 ADDRESS STROBE (ADS#)

This input, in conjunction with the READY# input indicates when the NPX bus-control logic may sample W/R# and the chip-select signals. Setup and hold times are referenced to CPUCLK2. This pin should be connected to the 386 DX CPU ADS# pin.

3.1.11 BUS READY INPUT (READY#)

This input indicates to the NPX when a 386 DX CPU bus cycle is to be terminated. It is used by the bus-control logic to trace bus activities. Bus cycles can be extended indefinitely until terminated by READY#. This input should be connected to the same signal that drives the 386 DX CPU READY# input. Setup and hold times are referenced to CPUCLK2.

3.1.12 READY OUTPUT (READYO#)

This pin is activated at such a time that write cycles are terminated after two clocks (except FLDENV and FRSTOR) and read cycles after three clocks. In configurations where no extra wait states are required, this pin must directly or indirectly drive the 386 DX CPU READY# input. Refer to section 3.4 "Bus Operation" for details. This pin is activated only during bus cycles that select the NPX. This signal is referenced to CPUCLK2.

3.1.13 STATUS ENABLE (STEN)

This pin serves as a chip select for the NPX. When inactive, this pin forces BUSY#, PEREQ, ERROR#, and READYO# outputs into floating state. D31-D0 are normally floating and leave floating state only if STEN is active and additional conditions are met. STEN also causes the chip to recognize its other chip-select inputs. STEN makes it easier to do on-board testing (using the overdrive method) of other chips in systems containing the NPX. STEN should be pulled up with a resistor so that it can be pulled down when testing. In boards that do not use on-board testing, STEN should be connected to V_{CC}. Setup and hold times are relative to CPUCLK2. Note that STEN must maintain the same setup and hold times as NPS1#, NPS2, and CMD0# (i.e. if STEN changes state during a 387 DX NPX bus cycle, it should change state during the same CLK period as the NPS1#, NPS2, and CMD0# signals).

3.1.14 NPX Select #1 (NPS1#)

When active (along with STEN and NPS2) in the first period of a 386 DX CPU bus cycle, this signal indicates that the purpose of the bus cycle is to commu-

nicate with the NPX. This pin should be connected directly to the 386 DX CPU M/IO# pin, so that the NPX is selected only when the 386 DX CPU performs I/O cycles. Setup and hold times are referenced to CPUCLK2.

3.1.15 NPX SELECT #2 (NPS2)

When active (along with STEN and NPS1#) in the first period of an 386 DX CPU bus cycle, this signal indicates that the purpose of the bus cycle is to communicate with the NPX. This pin should be connected directly to the 386 DX CPU A31 pin, so that the NPX is selected only when the 386 DX CPU uses one of the I/O addresses reserved for the NPX (800000F8 or 800000FC). Setup and hold times are referenced to CPUCLK2.

3.1.16 COMMAND (CMD0#)

During a write cycle, this signal indicates whether an opcode (CMD0# active) or data (CMD0# inactive) is being sent to the NPX. During a read cycle, it indicates whether the control or status register (CMD0# active) or a data register (CMD0# inactive) is being read. CMD0# should be connected directly to the A2 output of the 386 DX Microprocessor. Setup and hold times are referenced to CPUCLK2.

3.2 Processor Architecture

As shown by the block diagram on the front page, the NPX is internally divided into three sections: the bus control logic (BCL), the data interface and control unit, and the floating point unit (FPU). The FPU (with the support of the control unit which contains the sequencer and other support units) executes all numerics instructions. The data interface and control unit is responsible for the data flow to and from the FPU and the control registers, for receiving the instructions, decoding them, and sequencing the microinstructions, and for handling some of the administrative instructions. The BCL is responsible for the 386 DX CPU bus tracking and interface. The BCL is the only unit in the 387 DX NPX that must run synchronously with the 386 DX CPU; the rest of the NPX can run asynchronously with respect to the 386 DX Microprocessor.

3.2.1 BUS CONTROL LOGIC

The BCL communicates solely with the CPU using I/O bus cycles. The BCL appears to the CPU as a special peripheral device. It is special in two respects: the CPU initiates I/O automatically when it encounters ESC instructions, and the CPU uses reserved I/O addresses to communicate with the BCL. The BCL does not communicate directly with memory. The CPU performs all memory access, transferring input operands from memory to the NPX and transferring outputs from the NPX to memory.

3.2.2 DATA INTERFACE AND CONTROL UNIT

The data interface and control unit latches the data and, subject to BCL control, directs the data to the FIFO or the instruction decoder. The instruction decoder decodes the ESC instructions sent to it by the CPU and generates controls that direct the data flow in the FIFO. It also triggers the microinstruction sequencer that controls execution of each instruction. If the ESC instruction is FINIT, FCLEX, FSTSW, FSTSW AX, or FSTCW, the control executes it inde-

pendently of the FPU and the sequencer. The data interface and control unit is the one that generates the BUSY#, PEREQ and ERROR# signals that synchronize 387 DX NPX activities with the 386 DX CPU. It also supports the FPU in all operations that it cannot perform alone (e.g. exceptions handling, transcendental operations, etc.).

3.2.3 FLOATING POINT UNIT

The FPU executes all instructions that involve the register stack, including arithmetic, logical, transcendental, constant, and data transfer instructions. The data path in the FPU is 84 bits wide (68 significant bits, 15 exponent bits, and a sign bit) which allows internal operand transfers to be performed at very high speeds.

3.3 System Configuration

As an extension to the 386 DX Microprocessor, the 387 DX Math Coprocessor can be connected to the CPU as shown by Figure 3.3. A dedicated communi-

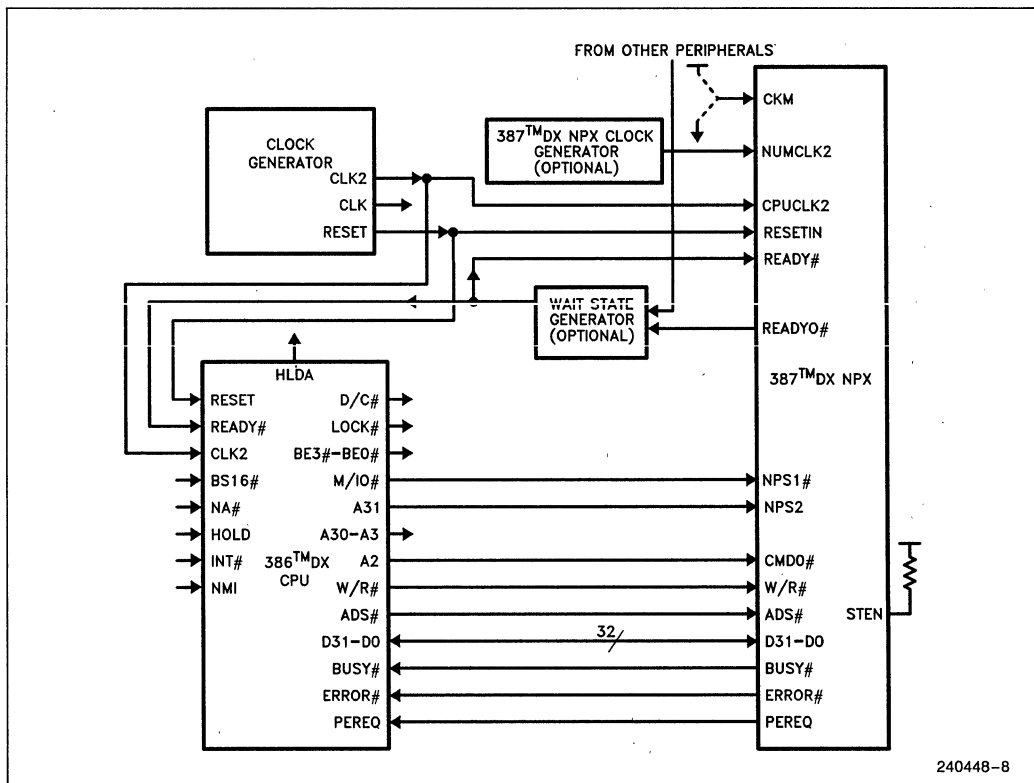


Figure 3.3. 386™ DX Microprocessor and 387™ DX Math Coprocessor System Configuration

Table 3.4. Bus Cycles Definition

STEN	NPS1#	NPS2	CMD0#	W/R#	Bus Cycle Type
0	x	x	x	x	NPX not selected and all outputs in floating state
1	1	x	x	x	NPX not selected
1	x	0	x	x	NPX not selected
1	0	1	0	0	CW or SW read from NPX
1	0	1	0	1	Opcode write to NPX
1	0	1	1	0	Data read from NPX
1	0	1	1	1	Data write to NPX

cation protocol makes possible high-speed transfer of opcodes and operands between the 386 DX CPU and 387 DX NPX. The 387 DX NPX is designed so that no additional components are required for interface with the 386 DX CPU. The 387 DX NPX shares the 32-bit wide local bus of the 386 DX CPU and most control pins of the 387 DX NPX are connected directly to pins of the 386 DX Microprocessor.

3.3.1 BUS CYCLE TRACKING

The ADS# and READY# signals allow the NPX to track the beginning and end of the 386 DX CPU bus cycles, respectively. When ADS# is asserted at the same time as the NPX chip-select inputs, the bus cycle is intended for the NPX. To signal the end of a bus cycle for the NPX, READY# may be asserted directly or indirectly by the NPX or by other bus-control logic. Refer to Table 3.4 for definition of the types of NPX bus cycles.

3.3.2 NPX ADDRESSING

The NPS1#, NPS2 and STEN signals allow the NPX to identify which bus cycles are intended for the NPX. The NPX responds only to I/O cycles when bit 31 of the I/O address is set. In other words, the NPX acts as an I/O device in a reserved I/O address space.

Because A₃₁ is used to select the NPX for data transfers, it is not possible for a program running on the 386 DX CPU to address the NPX with an I/O instruction. Only ESC instructions cause the 386 DX Microprocessor to communicate with the NPX. The 386 DX CPU BS16# input must be inactive during I/O cycles when A₃₁ is active.

3.3.3 FUNCTION SELECT

The CMD0# and W/R# signals identify the four kinds of bus cycle: control or status register read, data read, opcode write, data write.

3.3.4 CPU/NPX Synchronization

The pin pairs BUSY#, PEREQ, and ERROR# are used for various aspects of synchronization between the CPU and the NPX.

BUSY# is used to synchronize instruction transfer from the 386 DX CPU to the NPX. When the NPX recognizes an ESC instruction, it asserts BUSY#. For most ESC instructions, the 386 DX CPU waits for the NPX to deassert BUSY# before sending the new opcode.

The NPX uses the PEREQ pin of the 386 DX CPU to signal that the NPX is ready for data transfer to or from its data FIFO. The NPX does not directly access memory; rather, the 386 DX Microprocessor provides memory access services for the NPX. Thus, memory access on behalf of the NPX always obeys the rules applicable to the mode of the 386 DX CPU, whether the 386 DX CPU be in real-address mode or protected mode.

Once the 386 DX CPU initiates an NPX instruction that has operands, the 386 DX CPU waits for PEREQ signals that indicate when the NPX is ready for operand transfer. Once all operands have been transferred (or if the instruction has no operands) the 386 DX CPU continues program execution while the NPX executes the ESC instruction.

In 8086/8087 systems, WAIT instructions may be required to achieve synchronization of both commands and operands. In 80286/80287, 386 DX Microprocessor and 387 DX Math Coprocessor systems, WAIT instructions are required only for operand synchronization; namely, after NPX stores to memory (except FSTSW and FSTCW) or loads from memory. Used this way, WAIT ensures that the value has already been written or read by the NPX before the CPU reads or changes the value.

Once it has started to execute a numerics instruction and has transferred the operands from the 386 DX CPU, the NPX can process the instruction in parallel with and independent of the host CPU. When the NPX detects an exception, it asserts the ERROR# signal, which causes a 386 DX CPU interrupt.

3.3.5 SYNCHRONOUS OR ASYNCHRONOUS MODES

The internal logic of the 387 DX NPX (the FPU) can either operate directly from the CPU clock (synchronous mode) or from a separate clock (asynchronous mode). The two configurations are distinguished by the CKM pin. In either case, the bus control logic (BCL) of the NPX is synchronized with the CPU clock. Use of asynchronous mode allows the 386 DX CPU and the FPU section of the NPX to run at different speeds. In this case, the ratio of the frequency of NUMCLK2 to the frequency of CPUCLK2 must lie within the range 10:16 to 14:10. Use of synchronous mode eliminates one clock generator from the board design.

3.3.6 AUTOMATIC BUS CYCLE TERMINATION

In configurations where no extra wait states are required, READY# can be used to drive the 386 DX CPU READY# input. If this pin is used, it should be connected to the logic that ORs all READY outputs from peripherals on the 386 DX CPU bus. READY# is asserted by the NPX only during I/O cycles that select the NPX. Refer to section 3.4 "Bus Operation" for details.

3.4 Bus Operation

With respect to the bus interface, the 387 DX NPX is fully synchronous with the 386 DX Microprocessor. Both operate at the same rate, because each generates its internal CLK signal by dividing CPUCLK2 by two.

The 386 DX CPU initiates a new bus cycle by activating ADS#. The NPX recognizes a bus cycle, if, during the cycle in which ADS# is activated, STEN, NPS1#, and NPS2 are all activated. Proper operation is achieved if NPS1# is connected to the M/I/O# output of the 386 DX CPU, and NPS2 to the A31 output. The 386 DX CPU's A31 output is guaranteed to be inactive in all bus cycles that do not address the NPX (i.e. I/O cycles to other devices, interrupt acknowledge, and reserved types of bus cycles). System logic must not signal a 16-bit bus cycle via the 386 DX CPU BS16# input during I/O cycles when A31 is active.

During the CLK period in which ADS# is activated, the NPX also examines the W/R# input signal to determine whether the cycle is a read or a write cycle and examines the CMD0# input to determine whether an opcode, operand, or control/status register transfer is to occur.

The 387 DX NPX supports both pipelined and non-pipelined bus cycles. A nonpipelined cycle is one for which the 386 DX CPU asserts ADS# when no other NPX bus cycle is in progress. A pipelined bus cycle is one for which the 386 DX CPU asserts ADS# and provides valid next-address and control signals as soon as in the second CLK period after the ADS# assertion for the previous 386 DX CPU bus cycle. Pipelining increases the availability of the bus by at least one CLK period. The NPX supports pipelined bus cycles in order to optimize address pipelining by the 386 DX CPU for memory cycles.

Bus operation is described in terms of an abstract state machine. Figure 3.4 illustrates the states and state transitions for NPX bus cycles:

- T_I is the idle state. This is the state of the bus logic after RESET, the state to which bus logic returns after every nonpipelined bus cycle, and the state to which bus logic returns after a series of pipelined cycles.
- T_{RS} is the READY# sensitive state. Different types of bus cycle may require a minimum of one or two successive T_{RS} states. The bus logic remains in T_{RS} state until READY# is sensed, at which point the bus cycle terminates. Any number of wait states may be implemented by delaying READY#, thereby causing additional successive T_{RS} states.
- T_P is the first state for every pipelined bus cycle.

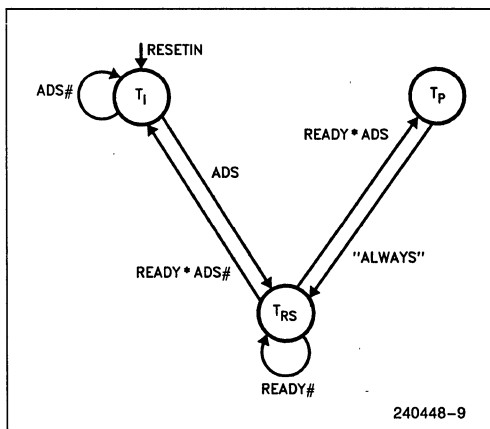


Figure 3.4. Bus State Diagram

The READY# output of the 387 DX NPX indicates when a bus cycle for the NPX may be terminated if no extra wait states are required. For all write cycles (except those for the instructions FLDENV and FRSTOR), READY# is always asserted in the first T_{RS} state, regardless of the number of wait states. For all read cycles and write cycles for FLDENV and FRSTOR, READY# is always asserted in the second T_{RS} state, regardless of the number of wait states. These rules apply to both pipelined and non-pipelined cycles. Systems designers must use READY# in one of the following ways:

1. Connect it (directly or through logic that ORs READY signals from other devices) to the READY# inputs of the 386 DX CPU and 387 DX NPX.
2. Use it as one input to a wait-state generator.

The following sections illustrate different types of NPX bus cycles.

Because different instructions have different amounts of overhead before, between, and after operand transfer cycles, it is not possible to represent in a few diagrams all of the combinations of successive operand transfer cycles. The following bus-cycle diagrams show memory cycles between NPX operand-transfer cycles. Note however that, during the instructions FLDENV, FSTENV, FSAVE, and FRSTOR, some consecutive accesses to the NPX do not have intervening memory accesses: For the timing relationship between operand transfer cycles and opcode write or other overhead activities, see Figure 3.8.

3.4.1 NONPIPELINED BUS CYCLES

Figure 3.5 illustrates bus activity for consecutive nonpipelined bus cycles.

3.4.1.1 Write Cycle

At the second clock of the bus cycle, the 387 DX NPX enters the T_{RS} (READY#-sensitive) state. During this state, the 387 DX NPX samples the READY# input and stays in this state as long as READY# is inactive.

In write cycles, the NPX drives the READY# signal for one CLK period beginning with the second CLK of the bus cycle; therefore, the fastest write cycle takes two CLK cycles (see cycle 2 of Figure 3.5). For the instructions FLDENV and FRSTOR, however, the NPX forces a wait state by delaying the activation of READY# to the second T_{RS} cycle (not shown in Figure 3.5).

When READY# is asserted the NPX returns to the idle state, in which ADS# could be asserted again by the 386 DX Microprocessor for the next cycle.

3.4.1.2 Read Cycle

At the second clock of the bus cycle, the NPX enters the T_{RS} state. See Figure 3.5. In this state, the NPX samples the READY# input and stays in this state as long as READY# is inactive.

At the rising edge of CLK in the second clock period of the cycle, the NPX starts to drive the D31–D0 outputs and continues to drive them as long as it stays in T_{RS} state.

In read cycles that address the NPX, at least one wait state must be inserted to insure that the 386 DX CPU latches the correct data. Since the NPX starts driving the system data bus only at the rising edge of CLK in the second clock period of the bus cycle, not enough time is left for the data signals to propagate and be latched by the 386 DX CPU at the falling edge of the same clock period. The NPX drives the READY# signal for one CLK period in the third CLK of the bus cycle. Therefore, if the READY# output is used to drive the 386 DX CPU READY# input, one wait state is inserted automatically.

Because one wait state is required for NPX reads, the minimum is three CLK cycles per read, as cycle 3 of Figure 3.5 shows.

When READY# is asserted the NPX returns to the idle state, in which ADS# could be asserted again by the 386 DX CPU for the next cycle. The transition from T_{RS} state to idle state causes the NPX to put the tristate D31–D0 outputs into the floating state, allowing another device to drive the system data bus.

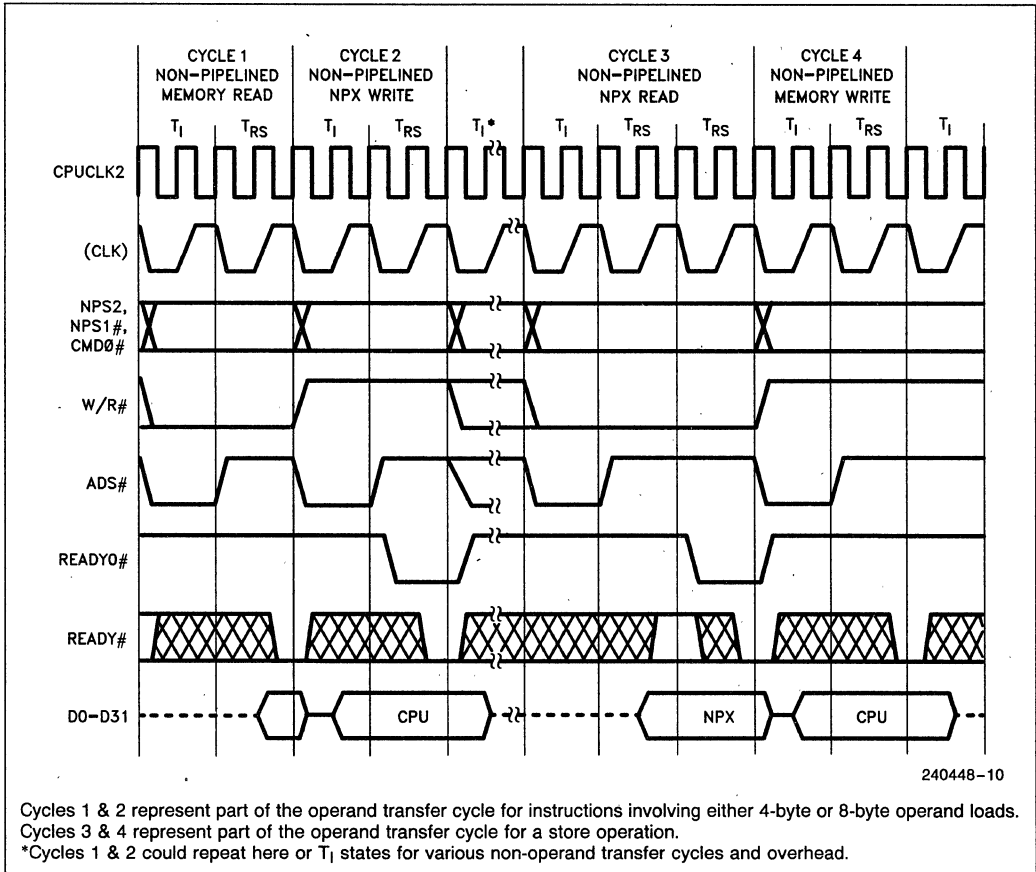


Figure 3.5. Nonpipelined Read and Write Cycles

3.4.2 PIPELINED BUS CYCLES

Because all the activities of the 387 DX NPX bus interface occur either during the T_{RS} state or during the transitions to or from that state, the only difference between a pipelined and a nonpipelined cycle is the manner of changing from one state to another. The exact activities in each state are detailed in the previous section "Nonpipelined Bus Cycles".

When the 386 DX CPU asserts ADS# before the end of a bus cycle, both ADS# and READY# are active during a T_{RS} state. This condition causes the NPX to change to a different state named T_p. The NPX activities in the transition from a T_{RS} state to a T_p state are exactly the same as those in the transition from a T_{RS} state to a T₁ state in nonpipelined cycles.

T_p state is metastable; therefore, one clock period later the NPX returns to T_{RS} state. In consecutive pipelined cycles, the NPX bus logic uses only T_{RS} and T_p states.

Figure 3.6 shows the fastest transition into and out of the pipelined bus cycles. Cycle 1 in this figure represents a nonpipelined cycle. (Nonpipelined write cycles with only one T_{RS} state (i.e. no wait states) are always followed by another nonpipelined cycle, because READY# is asserted before the earliest possible assertion of ADS# for the next cycle.)

Figure 3.7 shows the pipelined write and read cycles with one additional T_{RS} states beyond the minimum required. To delay the assertion of READY# requires external logic.

3.4.3 BUS CYCLES OF MIXED TYPE

When the 387 DX NPX bus logic is in the T_{RS} state, it distinguishes between nonpipelined and pipelined cycles according to the behavior of $ADS\#$ and $READY\#$. In a nonpipelined cycle, only $READY\#$ is activated, and the transition is from T_{RS} to idle state. In a pipelined cycle, both $READY\#$ and $ADS\#$ are active and the transition is first from T_{RS} state to T_P state then, after one clock period, back to T_{RS} state.

3.4.4 BUSY# AND PEREQ TIMING RELATIONSHIP

Figure 3.8 shows the activation of $BUSY\#$ at the beginning of instruction execution and its deactivation

after execution of the instruction is complete. When possible, the 387 DX NPX may deactivate $BUSY\#$ prior to the completion of the current instruction allowing the CPU to transfer the next instruction's opcode and operands. $PEREQ$ is activated in this interval. If $ERROR\#$ (not shown in the diagram) is ever asserted, it would occur at least six $CPUCLK2$ periods after the deactivation of $PEREQ$ and at least six $CPUCLK2$ periods before the deactivation of $BUSY\#$. Figure 3.8 shows also that $STEN$ is activated at the beginning of a bus cycle.

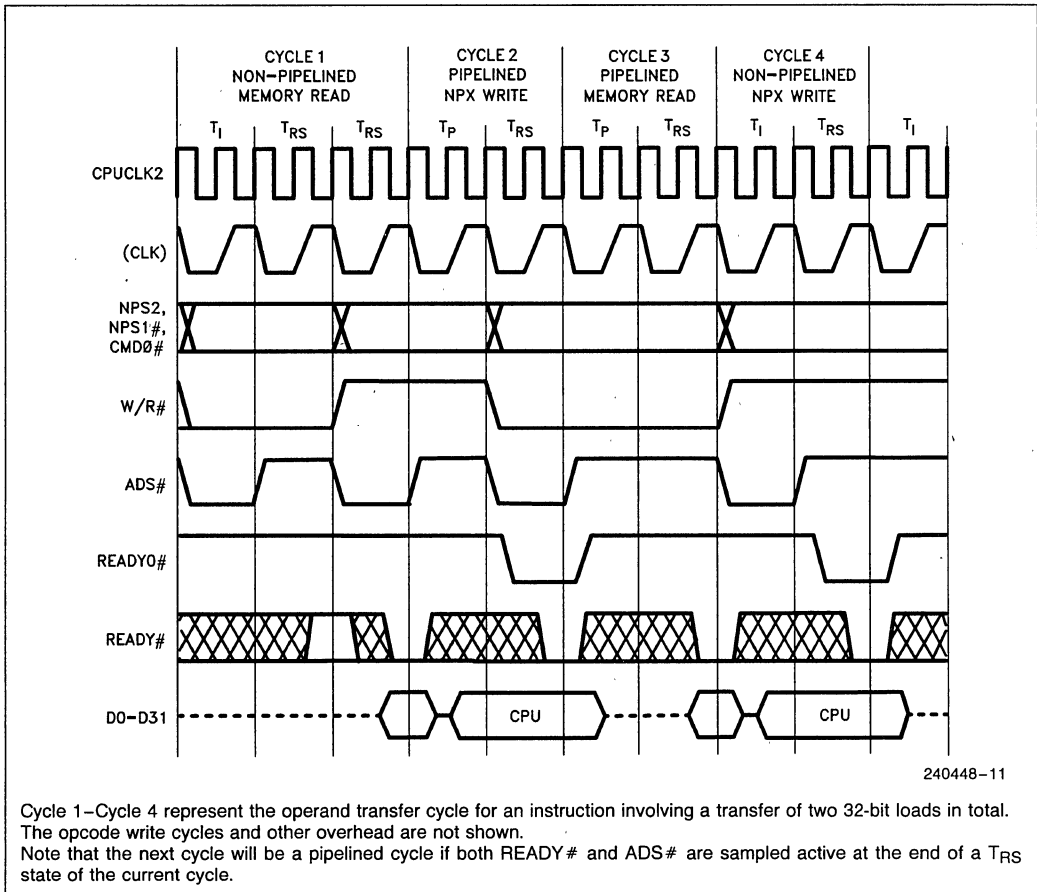


Figure 3.6. Fastest Transitions to and from Pipelined Cycles

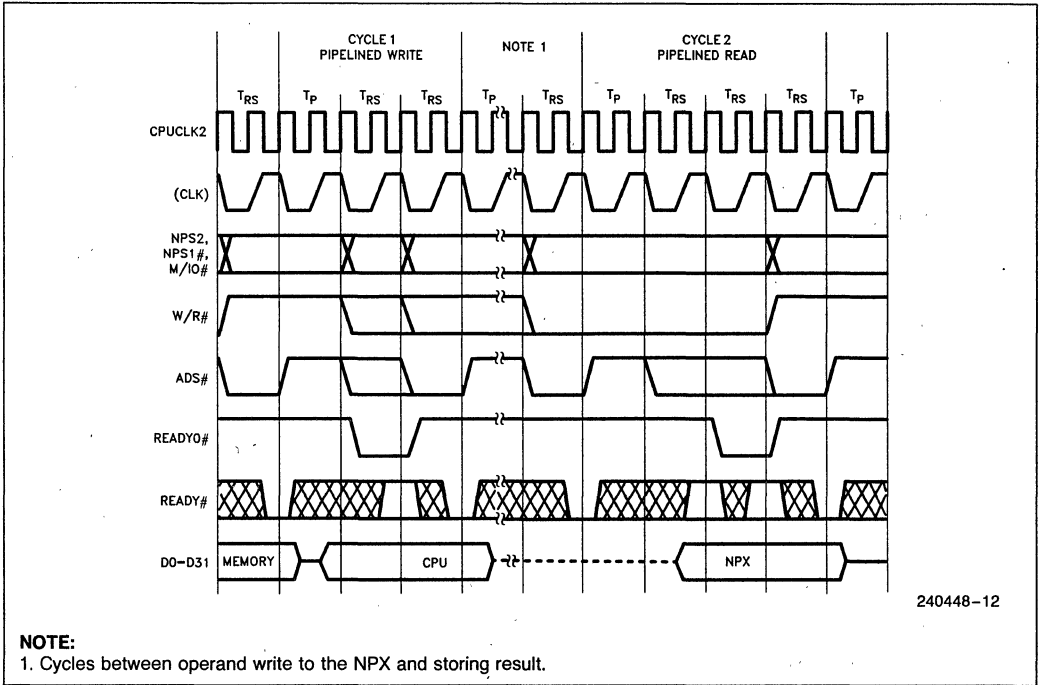


Figure 3.7. Pipelined Cycles with Wait States

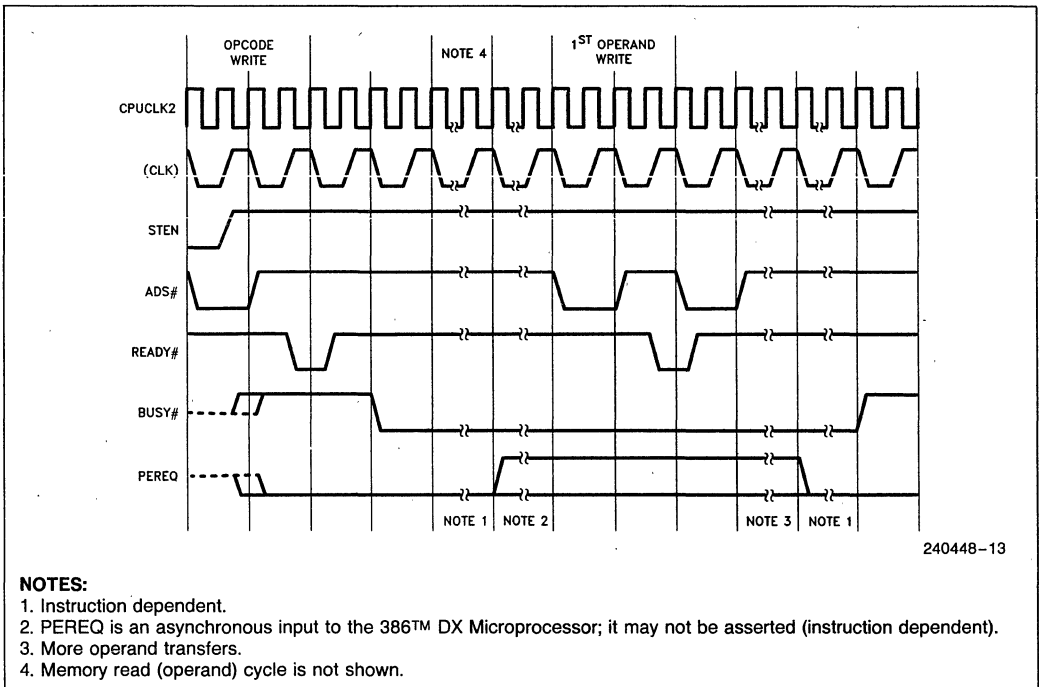


Figure 3.8. STEN, BUSY # and PEREQ Timing Relationship

4.0 ELECTRICAL DATA

4.1 Absolute Maximum Ratings*

Case Temperature T_C	
Under Bias	-65°C to +110°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin with	
Respect to Ground	-0.5 to $V_{CC} + 0.5V$
Power Dissipation	1.5W

*Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

NOTICE: Specifications contained within the following tables are subject to change.

4.2 D.C. Characteristics

Table 5.1. DC Specifications $T_C = 0^\circ$ to $85^\circ C$, $V_{CC} = 5V \pm 5\%$

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input LO Voltage	-0.3	+0.8	V	(Note 1)
V_{IH}	Input HI Voltage	2.0	$V_{CC} + 0.3$	V	(Note 1)
V_{CL}	CPUCLK2 Input LO Voltage	-0.3	+0.8	V	
V_{CH}	CPUCLK2 Input HI Voltage	3.7	$V_{CC} + 0.3$	V	
V_{OL}	Output LO Voltage		0.45	V	(Note 2)
V_{OH}	Output HI Voltage	2.4		V	(Note 3)
I_{CC}	Supply Current				
	NUMCLK2 = 32 MHz(4,6)		250	mA	I_{CC} typ. = 150 mA
	NUMCLK2 = 40 MHz(4,6)		310	mA	I_{CC} typ. = 190 mA
	NUMCLK2 = 50 MHz(4,6)		390	mA	I_{CC} typ. = 250 mA
	NUMCLK2 = 66.6 MHz(4,5)		250	mA	I_{CC} typ. = 150 mA
I_{LI}	Input Leakage Current		± 15	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	I/O Leakage Current		± 15	μA	$0.45V \leq V_O \leq V_{CC}$
C_{IN}	Input Capacitance		10	pF	fc = 1 MHz
C_O	I/O or Output Capacitance		12	pF	fc = 1 MHz
C_{CLK}	Clock Capacitance		15	pF	fc = 1 MHz

NOTES:

- This parameter is for all inputs, including NUMCLK2 but excluding CPUCLK2.
- This parameter is measured at I_{OL} as follows:
data = 4.0 mA
READYO# = 2.5 mA
ERROR#, BUSY#, PEREQ = 2.5 mA
- This parameter is measured at I_{OH} as follows:
data = 1.0 mA
READYO# = 0.6 mA
ERROR#, BUSY#, PEREQ = 0.6 mA
- I_{CC} is measured at steady state, maximum capacitive loading on the outputs, and worst-case DC level at the inputs; CPUCLK2 at the same frequency as NUMCLK2.
- I_{CC} specification for 387 DX-33 only (low power CHMOS IV process).
- I_{CC} specification for 387 DX-16, 20, 25 at corresponding maximum NUMCLK2 FREQ.

4.3 A.C. Characteristics

Table 4.2a. Combinations of Bus Interface and Execution Speeds

Functional Block	80387-16	80387-20	80387-25	80387DX-33
Bus Interface Unit (MHz)	16	20	25	33
Execution Unit (MHz)	16	20	25	33

Table 4.2b. Timing Requirements of the Execution Unit
 $T_C = 0^\circ\text{C to } +85^\circ\text{C}, V_{CC} = 5\text{V} \pm 5\%$

Pin	Symbol	Parameter	16 MHz		20 MHz		25 MHz		33 MHz		Test Conditions	Figure Reference
			Min (ns)	Max (ns)	Min (ns)	Max (ns)	Min (ns)	Max (ns)	Min (ns)	Max (ns)		
NUMCLK2	t1	Period	31.25	125	25	125	20	125	15	125	2.0V	4.1
NUMCLK2	t2a	High Time	9		8		7		6.25		2.0V	
NUMCLK2	t2b	High Time	5		5		4		4.5		3.7V	
NUMCLK2	t3a	Low Time	9		8		7		6.25		2.0V	
NUMCLK2	t3b	Low Time	7		6		5		4.5		0.8V	
NUMCLK2	t4	Fall Time		8		8		7			3.7V to 0.8V	
NUMCLK2	t5	Rise Time		8		8		7			0.8V to 8.7V	

Table 4.2c. Timing Requirements of the Bus Interface Unit
 $T_C = 0^\circ\text{C to } +85^\circ\text{C}, V_{CC} = 5\text{V} \pm 5\%$

 (All measurements made at 1.5V and $C_L = 50\text{ pF}$ unless otherwise specified)

Pin	Symbol	Parameter	16 MHz		20 MHz		25 MHz		33 MHz		Test Conditions	Figure Reference
			Min (ns)	Max (ns)	Min (ns)	Max (ns)	Min (ns)	Max (ns)	Min (ns)	Max (ns)		
CPUCLK2	t1	Period	31.25	125	25	125	20	125	15	125	2.0V	4.1
CPUCLK2	t2a	High Time	9		8		7		6.25		2.0V	
CPUCLK2	t2b	High Time	5		5		4		4.5		3.7V	
CPUCLK2	t3a	Low Time	9		8		7		6.25		2.0V	
CPUCLK2	t3b	Low Time	7		6		5		4.5		0.8V	
CPUCLK2	t4	Fall Time		8		8		7		4	3.7V to 0.8V	
CPUCLK2	t5	Rise Time		8		8		7		4	0.8V to 3.7V	
CPUCLK2/ NUMCLK2		Ratio	10/16	14/10	10/16	14/10	10/16	14/10	10/16	14/10		
READYO #	t7	Out Delay	3	34	3	31	3	24	3	17	$C_L = 75\text{ pF}\dagger$	4.2
READYO # (2)	t7	Out Delay	4	31	3	27	3	21	3	15	$C_L = 25\text{ pF}\dagger\dagger$	
PEREQ (1)	t7	Out Delay	5	34	5	34	4	33	4	25	$C_L = 75\text{ pF}\dagger$	
BUSY # (1)	t7	Out Delay	5	34	5	29	4	29	4	21	$C_L = 75\text{ pF}\dagger$	
BUSY # (1,2)	t7	Out Delay	N/A	N/A	N/A	N/A	4	27	4	19	$C_L = 25\text{ pF}\dagger\dagger$	
ERROR # (1)	t7	Out Delay	5	34	5	34	4	33	4	25	$C_L = 75\text{ pF}\dagger$	
D31-D0	t8	Out Delay	1	54	1	54	0	40	0	37	$C_L = 120\text{ pF}\dagger$	4.3
D31-D0	t10	Setup Time	11		11		11		8			
D31-D0	t11	Hold Time	11		11		11		8			
D31-D0 (3)	t12*	Float Time	6	33	6	27	5	24	3	19	$C_L = 120\text{ pF}\dagger$	
PEREQ (3)	t13*	Float Time	1	60	1	50	1	40	1	30	$C_L = 75\text{ pF}\dagger$	4.5
BUSY # (3)	t13*	Float Time	1	60	1	50	1	40	1	30	$C_L = 75\text{ pF}\dagger$	
ERROR # (3)	t13*	Float Time	1	60	1	50	1	40	1	30	$C_L = 75\text{ pF}\dagger$	
READYO # (3)	t13*	Float Time	1	60	1	50	1	40	1	30	$C_L = 75\text{ pF}\dagger$	

 *Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not tested.

 †For 25 MHz and 33 MHz, $C_L = 50\text{ pF}$

 ††For 33 MHz, $C_L = 50\text{ pF}$

Table 4.2c. Timing Requirements of the Bus Interface Unit (Continued)
T_C = 0°C to +85°C, V_{CC} = 5V ±5%
(All measurements made at 1.5V and C_L = 50 pF unless otherwise specified)

Pin	Symbol	Parameter	16 MHz		20 MHz		25 MHz		33 MHz		Figure Reference
			Min (ns)	Max (ns)	Min (ns)	Max (ns)	Min (ns)	Max (ns)	Max (ns)	Min (ns)	
ADS #	t14	Setup Time	25		20		15		13		4.3
ADS #	t15	Hold Time	5		5		4		4		
W/R #	t14	Setup Time	25		20		15		13		
W/R #	t15	Hold Time	5		5		4		4		
READY #	t16	Setup Time	20		11		8		7		
READY #	t17	Hold Time	4		4		4		4		
CMD0 #	t16	Setup Time	20		18		15		13		
CMD0 #	t17	Hold Time	2		2		4		4		
NPS1 #	t16	Setup Time	20		18		15		13		
NPS2											
NPS1 #	t17	Hold Time	2		2		4		4		
NPS2											
STEN	t16	Setup Time	20		20		14		13		
STEN	t17	Hold Time	2		2		2		2		
RESETIN	t18	Setup Time	13		12		10		5		4.4
RESETIN	t19	Hold Time	4		4		3		3		

NOTES:

- PEREQ, BUSY#, ERROR#
Out Delay 4 @ T_C = 0°C
 4 @ T_C = 85°C
- Not tested at 25 pF.
- Float delay is not tested. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude.

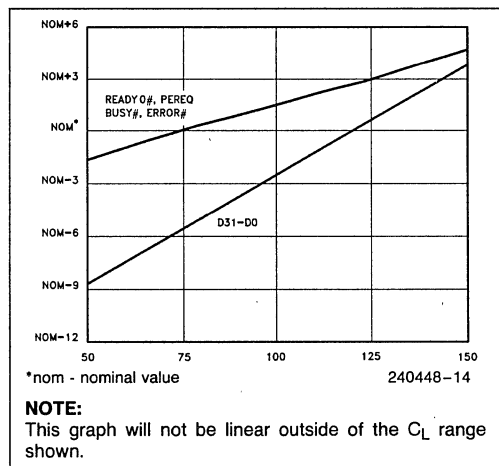


Figure 4.0a. Typical Output Valid Delay vs Load Capacitance at Max Operating Temperature

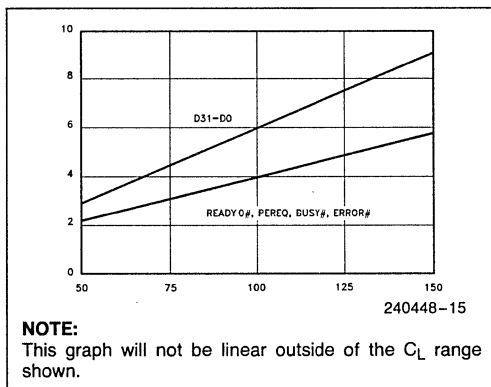
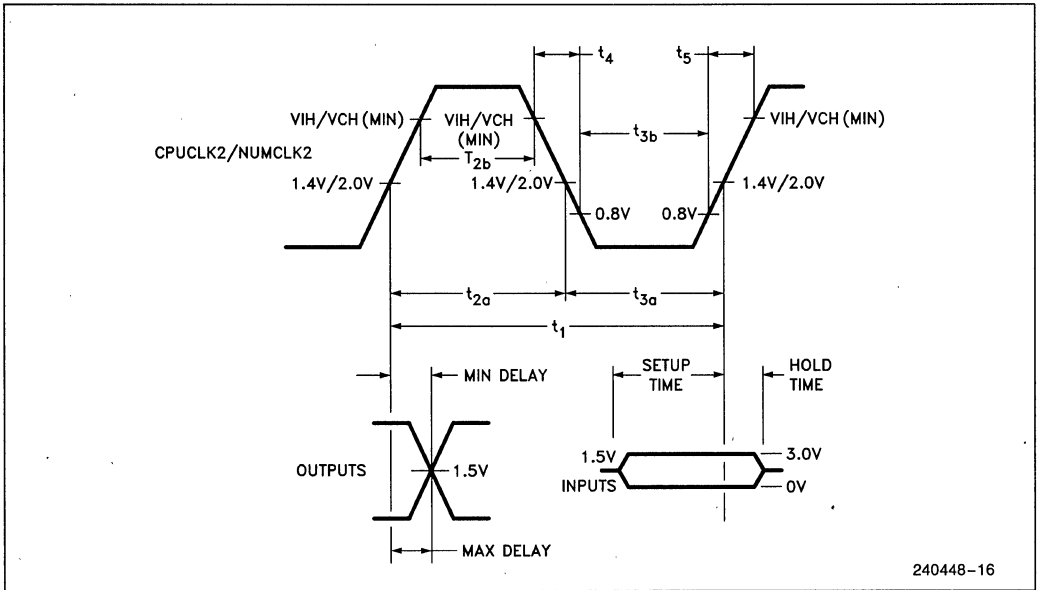
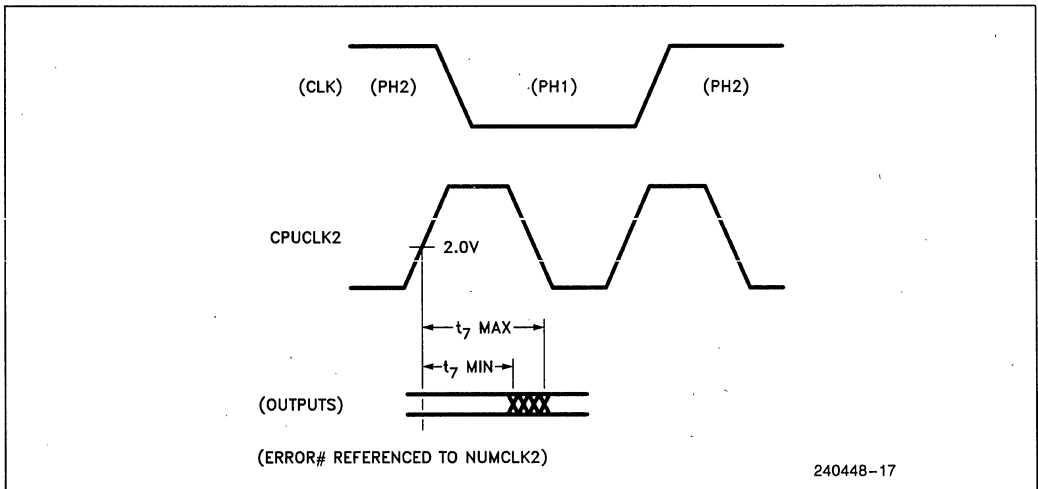


Figure 4.0b. Typical Output Rise Time vs Load Capacitance at Max Operating Temperature



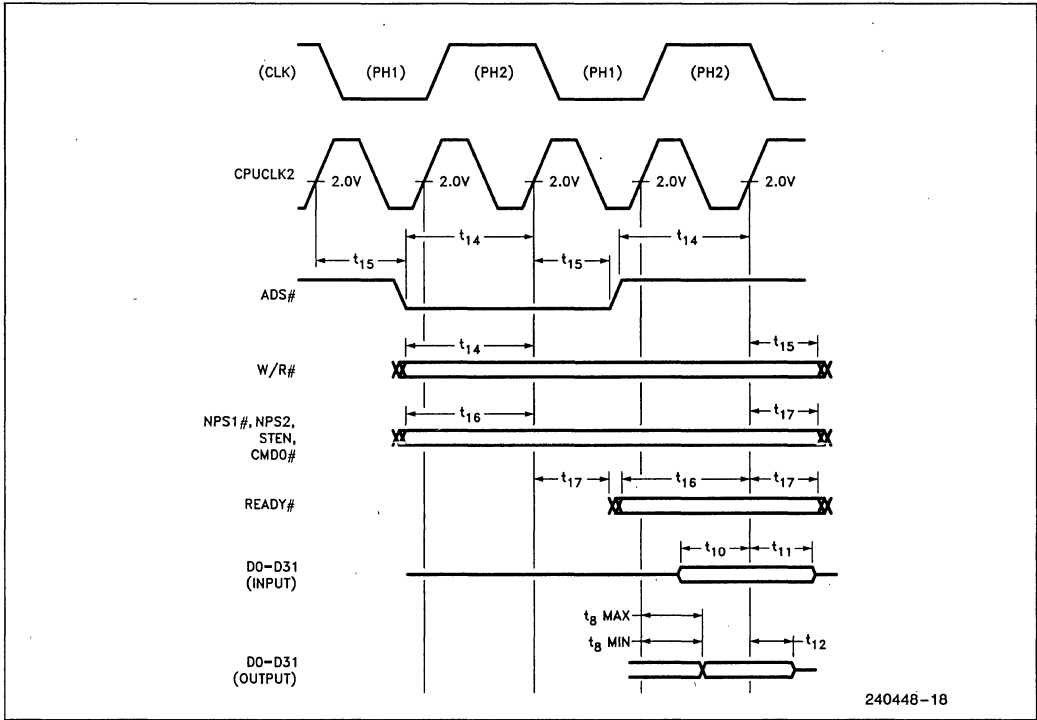
240448-16

Figure 4.1. CPUCLK2/NUMCLK2 Waveform and Measurement Points for Input/Output A.C. Specifications



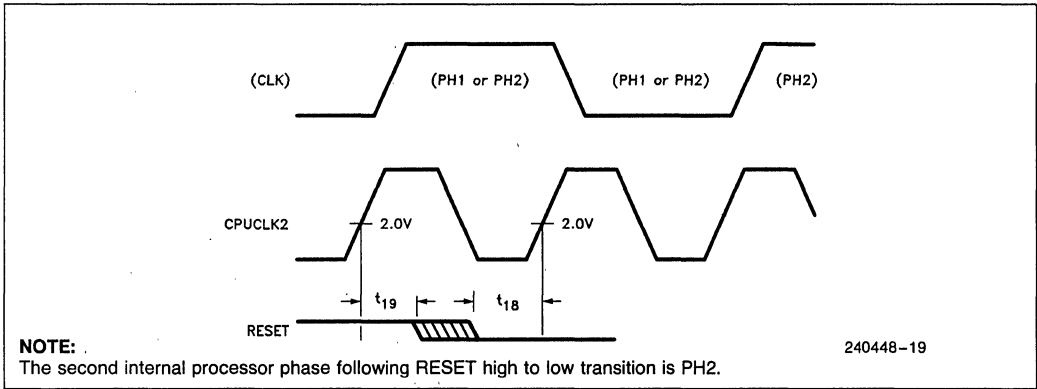
240448-17

Figure 4.2. Output Signals



240448-18

Figure 4.3. Input and I/O Signals

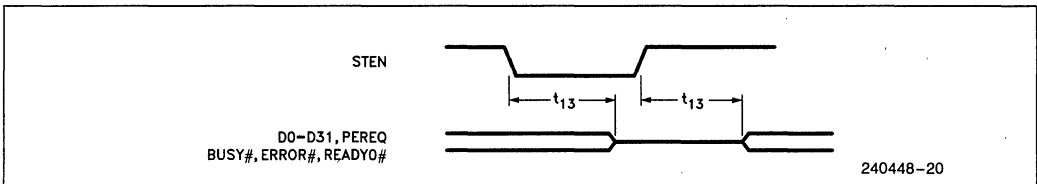


NOTE:

The second internal processor phase following RESET high to low transition is PH2.

240448-19

Figure 4.4. RESET Signal

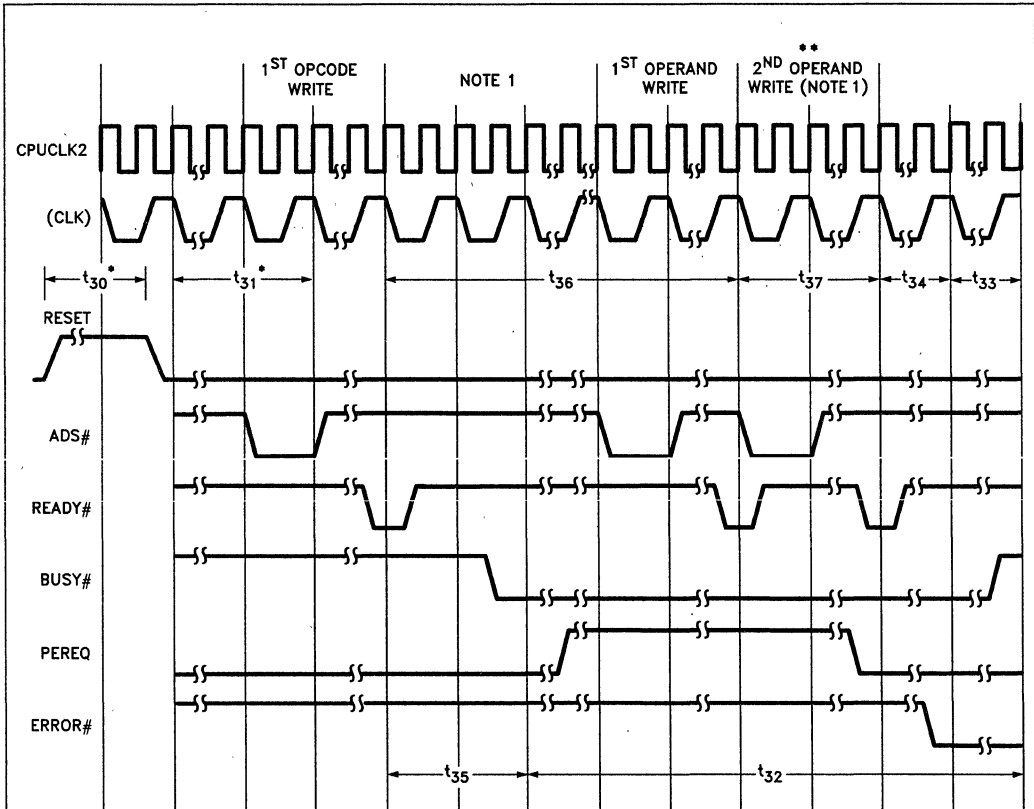


240448-20

Figure 4.5. Float from STEN

Table 4.3. Other Parameters

Pin	Symbol	Parameter	Min	Max	Units
RESETIN	t30	Duration	40		NUMCLK2
RESETIN	t31	RESETIN Inactive to 1st Opcode Write	50		NUMCLK2
BUSY #	t32	Duration	6		CPUCLK2
BUSY #, ERROR #	t33	ERROR # (In) Active to BUSY # Inactive	6		CPUCLK2
PEREQ, ERROR #	t34	PEREQ Inactive to ERROR # Active	6		CPUCLK2
READY #, BUSY #	t35	READY # Active to BUSY # Active	4	4	CPUCLK2
READY #	t36	Minimum Time from Opcode Write to Opcode/Operand Write	6		CPUCLK2
READY #	t37	Minimum Time from Operand Write to Operand Write	8		CPUCLK2



240448-21

* In NUMCLK2's
 ** or last operand

NOTE:
 1. Memory read (operand) cycle is not shown.

Figure 4.6. Other Parameters

		Instruction								Optional Fields		
		First Byte				Second Byte						
1	11011	OPA		1	MOD		1	OPB	R/M	SIB	DISP	
2	11011	MF			OPA	MOD		OPB		R/M	SIB	DISP
3	11011	d	P	OPA	1	1	OPB		ST(i)			
4	11011	0	0	1	1	1	1	OP				
5	11011	0	1	1	1	1	1	OP				
		15-11	10	9	8	7	6	5	4 3 2 1 0			

5.0 387™ DX NPX EXTENSIONS TO THE 386™ DX CPU INSTRUCTION SET

Instructions for the 387 DX NPX assume one of the five forms shown in the following table. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B, which identifies the ESCAPE class of instruction. Instructions that refer to memory operands specify addresses using the 386 DX CPU addressing modes.

OP = Instruction opcode, possible split into two fields OPA and OPB

MF = Memory Format
 00—32-bit real
 01—32-bit integer
 10—64-bit real
 11—16-bit integer

P = Pop
 0—Do not pop stack
 1—Pop stack after operation

ESC = 11011

d = Destination
 0—Destination is ST(0)
 1—Destination is ST(i)

R XOR d = 0—Destination (op) Source
 R XOR d = 1—Source (op) Destination

ST(i) = Register stack element *i*
 000 = Stack top
 001 = Second stack element
 .
 .
 .
 111 = Eighth stack element

MOD (Mode field) and R/M (Register/Memory specifier) have the same interpretation as the corresponding fields of the 386 DX Microprocessor instructions (refer to *386™ DX Microprocessor Programmer's Reference Manual*).

SIB (Scale Index Base) byte and DISP (displacement) are optionally present in instructions that have MOD and R/M fields. Their presence depends on the values of MOD and R/M, as for 386 DX Microprocessor instructions.

The instruction summaries that follow assume that the instruction has been prefetched, decoded, and is ready for execution; that bus cycles do not require wait states; that there are no local bus HOLD request delaying processor access to the bus; and that no exceptions are detected during instruction execution. If the instruction has MOD and R/M fields that call for both base and index registers, add one clock.

387™ DX NPX Extensions to the 386™ DX CPU Instruction Set

Instruction	Encoding			Clock Count Range			
	Byte 0	Byte 1	Optional Bytes 2-6	32-Bit Real	32-Bit Integer	64-Bit Real	16-Bit Integer
DATA TRANSFER							
FLD = Load ^a							
Integer/real memory to ST(0)	ESC MF 1	MOD 000 R/M	SIB/DISP	18	35-42	23	42
Long integer memory to ST(0)	ESC 111	MOD 101 R/M	SIB/DISP		43-54		
Extended real memory to ST(0)	ESC 011	MOD 101 R/M	SIB/DISP		43		
BCD memory to ST(0)	ESC 111	MOD 100 R/M	SIB/DISP		69-97		
ST(i) to ST(0)	ESC 001	11000 ST(i)			12		
FST = Store							
ST(0) to integer/real memory	ESC MF 1	MOD 010 R/M	SIB/DISP	43	62-76	44	63-76
ST(0) to ST(i)	ESC 101	11010 ST(i)			11		
FSTP = Store and Pop							
ST(0) to integer/real memory	ESC MF 1	MOD 011 R/M	SIB/DISP	43	62-76	44	63-76
ST(0) to long integer memory	ESC 111	MOD 111 R/M	SIB/DISP		65-82		
ST(0) to extended real	ESC 011	MOD 111 R/M	SIB/DISP		52		
ST(0) to BCD memory	ESC 111	MOD 110 R/M	SIB/DISP		134-190		
ST(0) to ST(i)	ESC 101	11011 ST(i)			11		
FXCH = Exchange							
ST(i) and ST(0)	ESC 001	11001 ST(i)			17		
COMPARISON							
FCOM = Compare							
Integer/real memory to ST(0)	ESC MF 0	MOD 010 R/M	SIB/DISP	25	45-52	27	58-62
ST(i) to ST(0)	ESC 000	11010 ST(i)			21		
FCOMP = Compare and pop							
Integer/real memory to ST	ESC MF 0	MOD 011 R/M	SIB/DISP	25	45-52	27	58-62
ST(i) to ST(0)	ESC 000	11011 ST(i)			21		
FCOMPP = Compare and pop twice							
ST(1) to ST(0)	ESC 110	1101 1001			21		
FTST = Test ST(0)							
	ESC 001	1110 0100			25		
FUCOM = Unordered compare							
	ESC 101	11100 ST(i)			21		
FUCOMP = Unordered compare and pop							
	ESC 101	11101 ST(i)			21		
FUCOMPP = Unordered compare and pop twice							
	ESC 010	1110 1001			21		
FXAM = Examine ST(0)							
	ESC 001	11100101			29-37		
CONSTANTS							
FLDZ = Load +0.0 into ST(0)	ESC 001	1110 1110			17		
FLD1 = Load +1.0 into ST(0)	ESC 001	1110 1000			22		
FLDPI = Load pi into ST(0)	ESC 001	1110 1011			36		
FLDL2T = Load log ₂ (10) into ST(0)	ESC 001	1110 1001			36		

Shaded areas indicate instructions not available in 8087/80287.

NOTE:

a. When loading single- or double-precision zero from memory, add 5 clocks.

387™ DX NPX Extensions to the 386™ DX CPU Instruction Set (Continued)

Instruction	Encoding			Clock Count Range			
	Byte 0	Byte 1	Optional Bytes 2-6	32-Bit Real	32-Bit Integer	64-Bit Real	16-Bit Integer
CONSTANTS (Continued)							
FLDL2E = Load $\log_2(e)$ into ST(0)	ESC 001	1110 1010			36		
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	ESC 001	1110 1100			35		
FLDLN2 = Load $\log_e(2)$ into ST(0)	ESC 001	1110 1101			38		
ARITHMETIC							
FADD = Add							
Integer/real memory with ST(0)	ESC MF 0	MOD 000 R/M	SIB/DISP	21-29	41-56	26-34	53-64
ST(i) and ST(0)	ESC d P 0	11000 ST(i)			18-26 ^b		
FSUB = Subtract							
Integer/real memory with ST(0)	ESC MF 0	MOD 10 R R/M	SIB/DISP	21-29	41-56	26-34	53-64 ^c
ST(i) and ST(0)	ESC d P 0	1110 R R/M			18-26 ^d		
FMUL = Multiply							
Integer/real memory with ST(0)	ESC MF 0	MOD 001 R/M	SIB/DISP	24-32	50-71	28-53	63-74
ST(i) and ST(0)	ESC d P 0	1100 1 R/M			22-50 ^e		
FDIV = Divide							
Integer/real memory with ST(0)	ESC MF 0	MOD 11 R R/M	SIB/DISP	85	107-114 ^f	91	120-124 ^g
ST(i) and ST(0)	ESC d P 0	1111 R R/M			80 ^h		
FSQRT1 = Square root	ESC 001	1111 1010			104-111		
FSCALE = Scale ST(0) by ST(1)	ESC 001	1111 1101			63-82		
FPREM = Partial remainder	ESC 001	1111 1000			60-140		
FPREM1 = Partial remainder (IEEE)	ESC 001	1111 0101			78-168		
FRNDINT = Round ST(0) to integer	ESC 001	1111 1100			48-62		
FXTRACT = Extract components of ST(0)	ESC 001	1111 0100			57-63		
FABS = Absolute value of ST(0)	ESC 001	1110 0001			21		
FCHS = Change sign of ST(0)	ESC 001	1110 0000			23-24		

Shaded areas indicate instructions not available in 8087/80287.

NOTES:

- b. Add 3 clocks to the range when d = 1.
- c. Add 1 clock to **each** range when R = 1.
- d. Add 3 clocks to the range when d = 0.
- e. typical = 52 (When d = 0, 46-54, typical = 49).
- f. Add 1 clock to the range when R = 1.
- g. 135-141 when R = 1.
- h. Add 3 clocks to the range when d = 1.
- i. $-0 \leq ST(0) \leq +\infty$.

387™ DX NPX Extensions to the 386™ DX CPU Instruction Set (Continued)

Instruction	Encoding			Clock Count Range
	Byte 0	Byte 1	Optional Bytes 2-6	
TRANSCENDENTAL				
FCOS^k = Cosine of ST(0)	ESC 001	1111 1111		122-680
FPTAN^k = Partial tangent of ST(0)	ESC 001	1111 0010		162-430 ⁱ
FPATAN = Partial arctangent	ESC 001	1111 0011		250-420
FSIN^k = Sine of ST(0)	ESC 001	1111 1110		121-680
FSINCOS^k = Sine and cosine of ST(0)	ESC 001	1111 1011		150-650
F2XM1^l = $2^{ST(0)} - 1$	ESC 001	1111 0000		167-410
FYL2XM^m = $ST(1) * \log_2(ST(0))$	ESC 001	1111 0001		99-436
FYL2XP1ⁿ = $ST(1) * \log_2(ST(0) + 1.0)$	ESC 001	1111 1001		210-447
PROCESSOR CONTROL				
FINIT = Initialize NPX	ESC 011	1110 0011		33
FSTSW AX = Store status word	ESC 111	1110 0000		13
FLDCW = Load control word	ESC 001	MOD 101 R/M	SIB/DISP	19
FSTCW = Store control word	ESC 101	MOD 111 R/M	SIB/DISP	15
FSTSW = Store status word	ESC 101	MOD 111 R/M	SIB/DISP	15
FCLEX = Clear exceptions	ESC 011	1110 0010		11
FSTENV = Store environment	ESC 001	MOD 110 R/M	SIB/DISP	103-104
FLDENV = Load environment	ESC 001	MOD 100 R/M	SIB/DISP	71
FSAVE = Save state	ESC 101	MOD 110 R/M	SIB/DISP	375-376
FRSTOR = Restore state	ESC 101	MOD 100 R/M	SIB/DISP	308
FINCSTP = Increment stack pointer	ESC 001	1111 0111		21
FDECSTP = Decrement stack pointer	ESC 001	1111 0110		22
FFREE = Free ST(i)	ESC 101	1100 0 ST(i)		18
FNOP = No operations	ESC 001	1101 0000		12

Shaded areas indicate instructions not available in 8087/80287.

NOTES:

j. These timings hold for operands in the range $|x| < \pi/4$. For operands not in this range, up to 76 additional clocks may be needed to reduce the operand.

k. $0 \leq |ST(0)| < 2^{63}$.

l. $-1.0 \leq ST(0) \leq 1.0$.

m. $0 \leq ST(0) < \infty$, $-\infty < ST(1) < +\infty$.

n. $0 \leq |ST(0)| < (2 - \text{SQRT}(2))/2$, $-\infty < ST(1) < +\infty$.

APPENDIX A COMPATIBILITY BETWEEN THE 80287 AND THE 8087

The 80286/80287 operating in Real-Address mode will execute 8086/8087 programs without major modification. However, because of differences in the handling of numeric exceptions by the 80287 NPX and the 8087 NPX, exception-handling routines *may* need to be changed.

This appendix summarizes the differences between the 80287 NPX and the 8087 NPX, and provides details showing how 8086/8087 programs can be ported to the 80286/80287.

1. The NPX signals exceptions through a dedicated ERROR# line to the 80286. The NPX error signal does not pass through an interrupt controller (the 8087 INT signal does). Therefore, any interrupt-controller-oriented instructions in numeric exception handlers for the 8086/8087 should be deleted.
2. The 8087 instructions FENI/FNENI and FDISI/FNDISI perform no useful function in the 80287. If the 80287 encounters one of these opcodes in its instruction stream, the instruction will effectively be ignored—none of the 80287 internal states will be updated. While 8086/8087 containing these instructions may be executed on the 80286/80287, it is unlikely that the exception-handling routines containing these instructions will be completely portable to the 80287.
3. Interrupt vector 16 must point to the numeric exception handling routine.
4. The ESC instruction address saved in the 80287 includes any leading prefixes before the ESC opcode. The corresponding address saved in the 8087 does not include leading prefixes.
5. In Protected-Address mode, the format of the 80287's saved instruction and address pointers is different than for the 8087. The instruction opcode is not saved in Protected mode—exception handlers will have to retrieve the opcode from memory if needed.
6. Interrupt 7 will occur in the 80286 when executing ESC instructions with either TS (task switched) or EM (emulation) of the 80286 MSW set (TS = 1 or EM = 1). If TS is set, then a WAIT instruction will also cause interrupt 7. An exception handler should be included in 80286/80287 code to handle these situations.
7. Interrupt 9 will occur if the second or subsequent words of a floating-point operand fall outside a segment's size. Interrupt 13 will occur if the starting address of a numeric operand falls outside a segment's size. An exception handler should be included in 80286/80287 code to report these programming errors.
8. Except for the processor control instructions, all of the 80287 numeric instructions are automatically synchronized by the 80286 CPU—the 80286 automatically tests the BUSY# line from the 80287 to ensure that the 80287 has completed its previous instruction before executing the next ESC instruction. No explicit WAIT instructions are required to assure this synchronization. For the 8087 used with 8086 and 8088 processors, explicit WAITs are required before each numeric instruction to ensure synchronization. Although 8086/8087 programs having explicit WAIT instructions will execute perfectly on the 80286/80287 without reassembly, these WAIT instructions are unnecessary.
9. Since the 80287 does not require WAIT instructions before each numeric instruction, the ASM286 assembler does not automatically generate these WAIT instructions. The ASM86 assembler, however, automatically precedes every ESC instruction with a WAIT instruction. Although numeric routines generated using the ASM86 assembler will generally execute correctly on the 80286/80287, reassembly using ASM286 may result in a more compact code image.

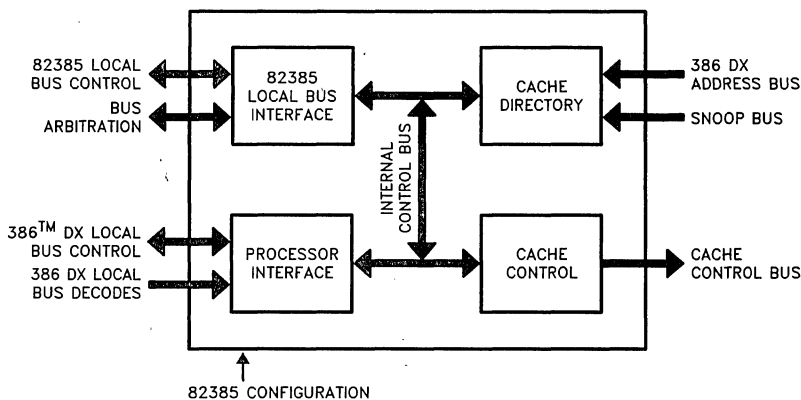
The processor control instructions for the 80287 may be coded using either a WAIT or No-WAIT form of mnemonic. The WAIT forms of these instructions cause ASM286 to precede the ESC instruction with a CPU WAIT instruction, in the identical manner as does ASM86.

82385 HIGH PERFORMANCE 32-BIT CACHE CONTROLLER

- **Improves 386™ DX System Performance**
 - Reduces Average CPU Wait States to Nearly Zero
 - Zero Wait State Read Hit
 - Zero Wait State Posted Memory Writes
 - Allows Other Masters to Access the System Bus More Readily
- **Hit Rates up to 99%**
- **Optimized as 386 DX Companion**
 - Simple 386 DX Interface
 - Part of 386 DX-Based Compute Engine Including 387™ DX Math Coprocessor and 82380 Integrated System Peripheral
 - 16 MHz, 20 MHz, 25 MHz, and 33 MHz Operation
- **Software Transparent**
- **Synchronous Dual Bus Architecture**
 - Bus Watching Maintains Cache Coherency
- **Maps Full 386 DX Address Space (4 Gigabytes)**
- **Flexible Cache Mapping Policies**
 - Direct Mapped or 2-Way Set Associative Cache Organization
 - Supports Non-Cacheable Memory Space
 - Unified Cache for Code and Data
- **Integrates Cache Directory and Cache Management Logic**
- **High Speed CHMOS III Technology**
- **132-Pin PGA Package**
- **132-Lead Plastic Quad Flat Pack (PQFP)**

The 82385 Cache Controller is a high performance 32-bit peripheral for the Intel386 Microprocessor. It stores a copy of frequently accessed code and data from main memory in a zero wait state local cache memory. The 82385 enables the 386 DX to run at its full potential by reducing the average number of CPU wait states to nearly zero. The dual bus architecture of the 82385 allows other masters to access system resources while the 386 DX operates locally out of its cache. In this situation, the 82385's "bus watching" mechanism preserves cache coherency by monitoring the system bus address lines at no cost to system or local throughput.

The 82385 is completely software transparent, protecting the integrity of system software. High performance and board savings are achieved because the 82385 integrates a cache directory and all cache management logic on one chip.



82385 Internal Block Diagram

290143-1

Intel386™, 386™ DX, 387™ DX are trademarks of Intel Corporation.

1.0 82385 FUNCTIONAL OVERVIEW

The 82385 Cache Controller is a high performance 32-bit peripheral for the Intel386 microprocessor. This chapter provides an overview of the 82385, and of the basic architecture and operation of an 386 DX CPU/82385 system.

1.1 82385 OVERVIEW

The main function of a cache memory system is to provide fast local storage for frequently accessed code and data. The cache system intercepts 386 DX memory references to see if the required data resides in the cache. If the data resides in the cache (a hit), it is returned to the 386 DX without incurring wait states. If the data is not cached (a miss), the reference is forwarded to the system and the data retrieved from main memory. An efficient cache will yield a high "hit rate" (the ratio of cache hits to total 386 DX accesses), such that the majority of accesses are serviced with zero wait states. The net effect is that the wait states incurred in a relatively infrequent miss are averaged over a large number of accesses, resulting in an average of nearly zero wait

states per access. Since cache hits are serviced locally, a processor operating out of its local cache has a much lower "bus utilization" which reduces system bus bandwidth requirements, making more bandwidth available to other bus masters.

The 82385 Cache Controller integrates a cache directory and all cache management logic required to support an external 32 Kbyte cache. The cache directory structure is such that the entire physical address range of the 386 DX (4 Gigabytes) is mapped into the cache. Provision is made to allow areas of memory to be set aside as non-cacheable. The user has two cache organization options: direct mapped and 2-way set associative. Both provide the high hit rates necessary to make a large, relatively slow main memory array look like a fast, zero wait state memory to the 386 DX.

1.2 SYSTEM OVERVIEW I: BUS STRUCTURE

A good grasp of the bus structure of a 386 DX CPU/82385 system is essential in understanding both the 82385 and its role in an 386 DX system. The following is a description of this structure.

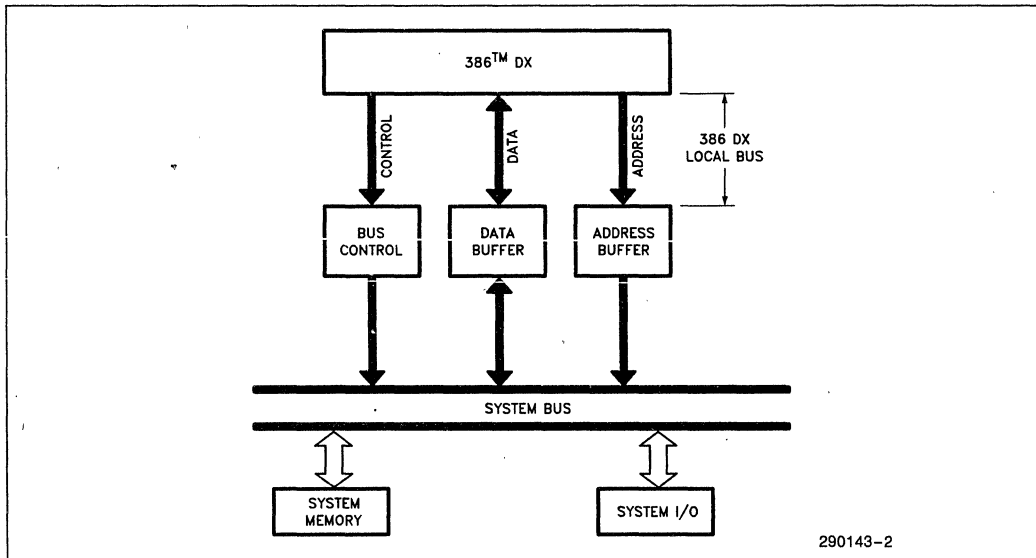


Figure 1-1. 386 DX System Bus Structure

1.2.1 386 DX Local Bus/82385 Local Bus/System Bus

Figure 1-1 depicts the bus structure of a typical 386 DX system. The "386 DX Local Bus" consists of the physical 386 DX address, data, and control busses. The local address and data busses are buffered and/or latched to become the "system" address and data busses. The local control bus is decoded by bus control logic to generate the various system bus read and write commands.

The addition of an 82385 Cache Controller causes a separation of the 386 DX bus into two distinct busses: the actual 386 DX local bus and the "82385 Local Bus" (Figure 1-2). The 82385 local bus is designed to look like the front end of an 386 DX by providing 82385 local bus equivalents to all appropriate 386 DX signals. The system ties to this "386 DX-like" front end just as it would to an actual 386 DX. The 386 DX simply sees a fast system bus, and the system sees a 386 DX front end with low bus bandwidth requirements. The cache subsystem is transparent to both. Note that the 82385 local bus is not simply a buffered version of the 386 DX bus, but rather is distinct from, and able to operate in parallel with the 386 DX bus. Other masters residing on either the 82385 local bus or system bus are free to manage system resources while the 386 DX operates out of its cache.

1.2.2 Bus Arbitration

The 82385 presents the "386 DX-like" interface which is called the 82385 local bus. Whereas the 386 DX provides a Hold Request/Hold Acknowledge bus arbitration mechanism via its HOLD and HLDA pins, the 82385 provides an equivalent mechanism via its BHOLD and BHLDA pins. (These signals are described in Section 3.7.) When another master requests the 82385 local bus, it issues the request to the 82385 via BHOLD. Typically, at the end of the current 82385 local bus cycle, the 82385 will release the 82385 local bus and acknowledge the request via BHLDA. The 386 DX is of course free to continue operating on the 386 DX local bus while another master owns the 82385 local bus.

1.2.3 Master/Slave Operation

The above 82385 local bus arbitration discussion is true when the 82385 is programmed for "Master" mode operation. The user can, however, configure the 82385 for "Slave" mode operation. (Programming is done via a hardware strap option.) The roles of BHOLD and BHLDA are reversed for an 82385 in slave mode; BHOLD is now an output indicating a request to control the bus, and BHLDA is an input indicating that a request has been granted. An 82385 programmed in slave mode drives the 82385 local bus only when it has requested and subsequently been granted bus control. This allows multiple 386 DX CPU/82385 subsystems to reside on the same 82385 local bus (Figure 1-3).

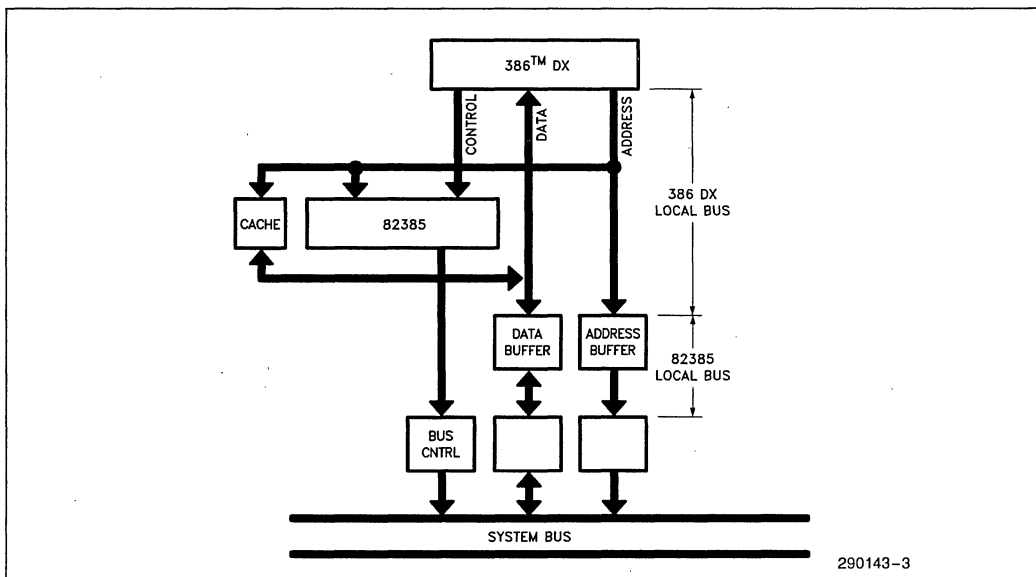


Figure 1-2. 386™ DX CPU/82385 System Bus Structure

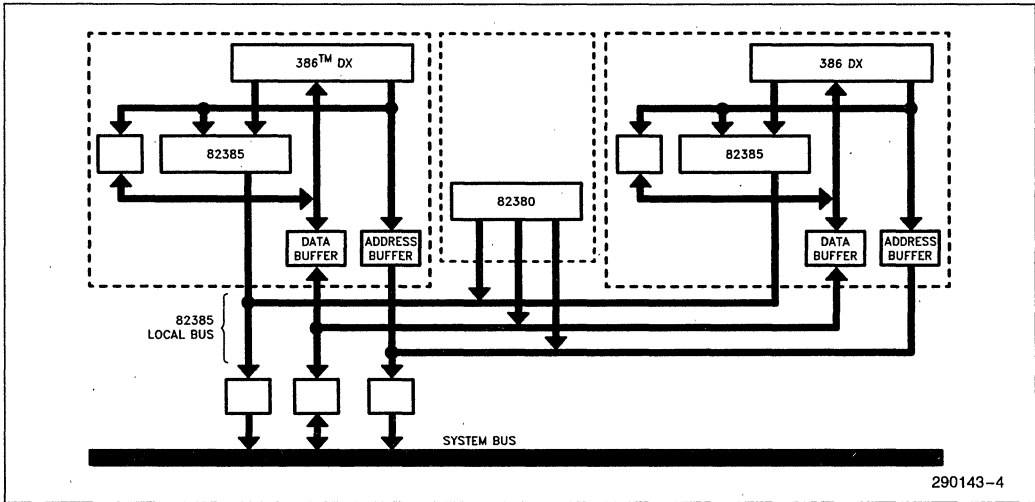


Figure 1-3. Multi-Master/Multi-Cache Environment

1.2.4 Cache Coherency

Ideally, a cache contains a copy of the most heavily used portions of main memory. To maintain cache "coherency" is to make sure that this local copy is identical to main memory. In a system where multiple masters can access the same memory, there is always a risk that one master will alter the contents of a memory location that is duplicated in the local cache of another master. (The cache is said to contain "stale" data.) One rather restrictive solution is to not allow cache subsystems to cache shared memory. Another simple solution is to flush the cache anytime another master writes to system memory. However, this can seriously degrade system performance as excessive cache flushing will reduce the hit

rate of what may otherwise be a highly efficient cache.

The 82385 preserves cache coherency via "bus watching" (also called snooping), a technique that neither impacts performance nor restricts memory mapping. An 82385 that is not currently bus master monitors system bus cycles, and when a write cycle by another master is detected (a snoop), the system address is sampled and used to see if the referenced location is duplicated in the cache. If so (a snoop hit), the corresponding cache entry is invalidated, which will force the 386 DX to fetch the up-to-date data from main memory the next time it accesses this modified location. Figure 1-4 depicts the general form of bus watching.

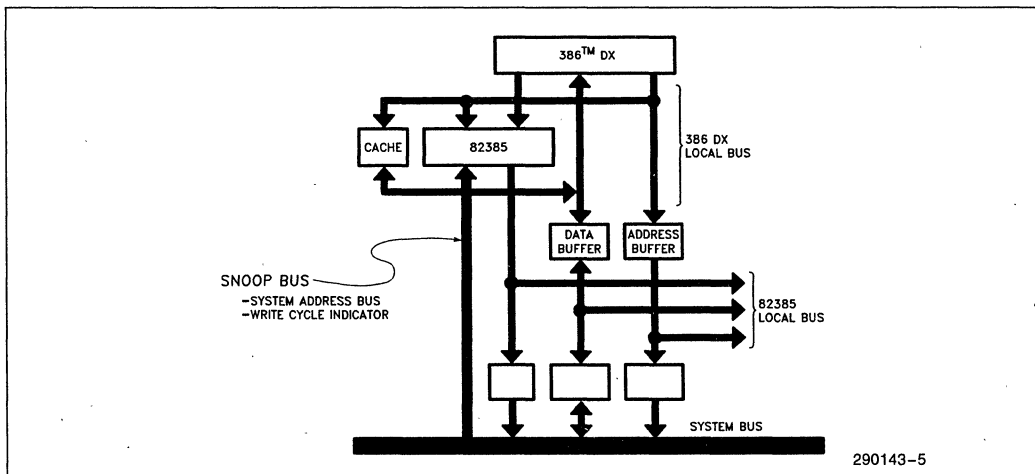


Figure 1-4. 82385 Bus Watching—Monitor System Bus Write Cycles

1.3 SYSTEM OVERVIEW II: BASIC OPERATION

This discussion is an overview of the basic operation of an 386 DX CPU/82385 system. Items discussed include the 82385's response to all 386 DX cycles, including interrupt acknowledges, halts, and shut-downs. Also discussed are non-cacheable and local accesses.

1.3.1 386 DX Memory Code and Data Read Cycles

1.3.1.1 READ HITS

When the 386 DX initiates a memory code or data read cycle, the 82385 compares the high order bits of the 386 DX address bus with the appropriate addresses (tags) stored in its on-chip directory. (The directory structure is described in Chapter 2.) If the 82385 determines that the requested data is in the cache, it issues the appropriate control signals that direct the cache to drive the requested data onto the 386 DX data bus, where it is read by the 386 DX. The 82385 terminates the 386 DX cycle without inserting any wait states.

1.3.1.2 READ MISSES

If the 82385 determines that the requested data is not in the cache, the request is forwarded to the 82385 local bus and the data retrieved from main memory. As the data returns from main memory, it is directed to the 386 DX and also written into the cache. Concurrently, the 82385 updates the cache directory such that the next time this particular piece of information is requested by the 386 DX, the 82385 will find it in the cache and return it with zero wait states.

The basic unit of transfer between main memory and cache memory in a cache subsystem is called the line size. In an 82385 system, the line size is one 32-bit aligned doubleword. During a read miss, all four 82385 local bus byte enables are active. This ensures that a full 32-bit entry is written into the cache. (The 386 DX simply ignores what it did not request.) In any other type of 386 DX cycle that is forwarded to the 82385 local bus, the logic levels of the 386 DX byte enables are duplicated on the 82385 local bus.

The 82385 does not actively fetch main memory data independently of the 386 DX. The 82385 is essentially a passive device which only monitors the address bus and activates control signals. The read miss is the only mechanism by which main memory data is copied into the cache and validated in the cache directory.

In an isolated read miss, the number of wait states seen by the 386 DX is that required by the system memory to respond with data plus the cache comparison cycle (hit/miss decision). The cache system must determine that the cycle is a miss before it can begin the system memory access. However, since misses most often occur consecutively, the 82385 will begin 386 DX address pipelined cycles to effectively "hide" the comparison cycle beyond the first miss (refer to Section 4.1.3).

The 82385 can execute a main memory access on the 82385 local bus only if it currently owns the bus. If not, an 82385 in master mode will run the cycle after the current master releases the bus. An 82385 in slave mode will issue a hold request, and will run the cycle as soon as the request is acknowledged. (This is true for any read or write cycle that needs to run on the 82385 local bus.)

1.3.2 386 DX Memory Write Cycles

The 82385's "posted write" capability allows the majority of 386 DX memory write cycles to run with zero wait states. The primary memory update policy implemented in a posted write is the traditional cache "write through" technique, which implies that main memory is always updated in any memory write cycle. If the referenced location also happens to reside in the cache (a write hit), the cache is updated as well.

Beyond this, a posted write latches the 386 DX address, data, and cycle definition signals, and the 386 DX local bus cycle is terminated without any wait states, even though the corresponding 82385 local bus cycle is not yet completed, or perhaps not even started. A posted write is possible because the 82385's bus state machine, which is almost identical to the 386 DX bus state machine, is able to run 82385 local bus cycles independently of the 386 DX. The only time the 386 DX sees write cycle wait states is when a previously latched (posted) write has not yet been completed on the 82385 local bus or during an I/O write (which is not posted). A 386 DX write can be posted even if the 82385 does not currently own the 82385 local bus. In this case, an 82385 in master mode will run the cycle as soon as the current master releases the bus, and an 82385 in slave mode will request the bus and run the cycle when the request is acknowledged. The 386 DX is free to continue operating out of its cache (on the 386 DX local bus) during this time.

1.3.3 Non-Cacheable Cycles

Non-cacheable cycles fall into one of two categories: cycles decoded as non-cacheable, and cycles

that are by default non-cacheable according to the 82385's design. All non-cacheable cycles are forwarded to the 82385 local bus. Non-cacheable cycles have no effect on the cache or cache directory.

The 82385 allows the system designer to define areas of main memory as non-cacheable. The 386 DX address bus is decoded and the decode output is connected to the 82385's non-cacheable access (NCA#) input. This decoding is done in the first 386 DX bus state in which the non-cacheable cycle address becomes available. Non-cacheable read cycles resemble cacheable read miss cycles, except that the cache and cache directory are unaffected. NCA defined non-cacheable writes, like most writes, are posted.

The 82385 defines certain cycles as non-cacheable without using its non-cacheable access input. These include I/O cycles, interrupt acknowledge cycles, and halt/shutdown cycles. I/O reads and interrupt acknowledge cycles execute as any other non-cacheable read. I/O write cycles are not posted. The 386 DX is not allowed to continue until a ready signal is returned from the system. Halt/Shutdown cycles are posted. During a halt/shutdown condition, the 82385 local bus duplicates the behavior of the 386 DX, including the ability to recognize and respond to a B HOLD request. (The 82385's bus watching mechanism is functional in this condition.)

1.3.3.1 16-BIT MEMORY SPACE

The 82385 does not cache 16-bit memory space (as decoded by the 386 DX BS16# input), but does make provisions to handle 16-bit space as non-cacheable. (There is no 82385 equivalent to the 386 DX BS16# input.) In a system without an 82385, the 386 DX BS16# input need not be asserted until the last state of a 16-bit cycle for the 386 DX to recognize it as such (unless NA# is sampled active earlier in the cycle.) The 82385, however, needs this information earlier, specifically at the end of the first 386 DX bus state in which the address of the 16-bit cycle becomes available. The result is that in a system without an 82385, 16-bit devices can inform the 386 DX that they are 16-bit devices "on the fly," while in

a system with an 82385, devices decoded as 16-bit (using the 386 DX BS16#) must be located in address space set aside for 16-bit devices. If 16-bit space is decoded according to 82385 guidelines (as described later in the data sheet), then the 82385 will handle 16-bit cycles just like the 386 DX does, including effectively locking the two halves of a non-aligned 16-bit transfer from interruption by another master.

1.3.4 386 DX Local Bus Cycles

386 DX Local Bus Cycles are accesses to resources on the 386 DX local bus other than to the 82385 itself. The 82385 simply ignores these accesses: they are neither forwarded to the system nor do they affect the cache. The designer sets aside memory and/or I/O space for local resources by decoding the 386 DX address bus and feeding the decode to the 82385's local bus access (LBA#) input. The designer can also decode the 386 DX cycle definition signals to keep specific 386 DX cycles from being forwarded to the system. For example, a multi-processor design may wish to capture and remedy a 386 DX shutdown locally without having it detected by the rest of the system. Note that in such a design, the local shutdown cycle must be terminated by local bus control logic. The 387 Math Coprocessor is considered a 386 DX local bus resource, but it need not be decoded as such by the user since the 82385 is able to internally recognize 387 accesses via the M/I/O# and A31 pins.

1.3.5 Summary of 82385 Response to All 386 DX Cycles

Table 1-1 summarizes the 82385 response to all 386 DX bus cycles, as conditioned by whether or not the cycle is decoded as local or non-cacheable. The table describes the impact of each cycle on the cache and on the cache directory, and whether or not the cycle is forwarded to the 82385 local bus. Whenever the 82385 local bus is marked "IDLE", it implies that this bus is available to other masters.

Table 1-1. 82385 Response to 386 DX Cycles

386 DX Bus Cycle Definition				82385 Response when Decoded as Cacheable				82385 Response when Decoded as Non-Cacheable			82385 Response when Decoded as an 386 DX Local Bus Access		
M/IO#	D/C#	W/R#	386 DX Cycle		Cache	Cache Directory	82385 Local Bus	Cache	Cache Directory	82385 Local Bus	Cache	Cache Directory	82385 Local Bus
0	0	0	INT ACK	N/A	—	—	INT ACK	—	—	INT ACK	—	—	IDLE
0	0	1	UNDEFINED	N/A			UNDEFINED			UNDEFINED			IDLE
0	1	0	I/O READ	N/A	—	—	I/O READ	—	—	I/O READ	—	—	IDLE
0	1	1	I/O WRITE	N/A	—	—	I/O WRITE	—	—	I/O WRITE	—	—	IDLE
1	0	0	MEM CODE READ	HIT	CACHE READ	—	IDLE	—	—	MEM CODE READ	—	—	IDLE
				MISS	CACHE WRITE	DATA VALIDATION	MEM CODE READ						
1	0	1	HALT/SHUTDOWN	N/A	—	—	HALT/SHUTDOWN	—	—	HALT/SHUTDOWN	—	—	IDLE
1	1	0	MEM DATA READ	HIT	CACHE READ	—	IDLE	—	—	MEM DATA READ	—	—	IDLE
				MISS	CACHE WRITE	DATA VALIDATION	MEM DATA READ						
1	1	1	MEM DATA WRITE	HIT	CACHE WRITE	—	MEM DATA WRITE	—	—	MEM DATA WRITE	—	—	IDLE
				MISS	—	—	MEM DATA WRITE						

NOTES:

- A dash (—) indicates that the cache and cache directory are unaffected. This table does not reflect how an access affects the LRU bit.
- An "IDLE" 82385 Local Bus implies that this bus is available to other masters.
- The 82385's response to 80387 accesses is the same as when decoded as an 386 DX Local Bus access.
- The only other operations that affect the cache directory are:
 1. RESET or Cache Flush—all tag valid bits cleared.
 2. Snoop Hit—corresponding line valid bit cleared.

1.3.6 Bus Watching

As previously discussed, the 82385 "qualifies" an 386 DX bus cycle in the first bus state in which the address and cycle definition signals of the cycle become available. The cycle is qualified as read or write, cacheable or non-cacheable, etc. Cacheable cycles are further classified as hit or miss according to the results of the cache comparison, which accesses the 82385 directory and compares the appropriate directory location (tag) to the current 386 DX address. If the cycle turns out to be non-cacheable or a 386 DX local bus access, the hit/miss decision is ignored. The cycle qualification requires one 386 DX state. Since the fastest 386 DX access is two states, the second state can be used for bus watching.

When the 82385 does not own the system bus, it monitors system bus cycles. If another master writes into main memory, the 82385 latches the system address and executes a cache look-up to see if the altered main memory location resides in the cache. If so (a snoop hit), the cache entry is marked invalid in the cache directory. Since the directory is at most only being used every other state to qualify 386 DX accesses, snoop look-ups are interleaved between 386 DX local bus look-ups. The cache directory is time multiplexed between the 386 DX address and the latched system address. The result is that all snoops are caught and serviced without slowing down the 386 DX, even when running zero wait state hits on the 386 DX local bus.

1.3.7 Cache Flush

The 82385 offers a cache flush input. When activated, this signal causes the 82385 to invalidate all data which had previously been cached. Specifically,

all tag valid bits are cleared. (Refer to the 82385 directory structure in Chapter 2.) Therefore, the cache is empty and subsequent cycles are misses until the 386 DX begins repeating the new accesses (hits). The primary use of the FLUSH input is for diagnostics and multi-processor support.

NOTE:

The use of this pin as a coherency mechanism may impact software transparency.

2.0 82385 CACHE ORGANIZATION

The 82385 supports two cache organizations: a simple direct mapped organization and a slightly more complex, higher performance two way set associative organization. The choice is made by strapping an 82385 input (2W/D#) either high or low. This chapter describes the structure and operation of both organizations.

2.1 DIRECT MAPPED CACHE

2.1.1 Direct Mapped Cache Structure and Terminology

Figure 2-1 depicts the relationship between the 82385's internal cache directory, the external cache memory, and the 386 DX's 4 Gigabyte physical address space. The 4 Gigabytes can conceptually be thought of as cache "pages" each being 8K doublewords (32 Kbytes) deep. The page size matches the cache size. The cache can be further divided into 1024 (0 thru 1023) sets of eight doublewords (8 x 32 bits). Each 32-bit doubleword is called a "line." The unit of transfer between the main memory and cache is one line.

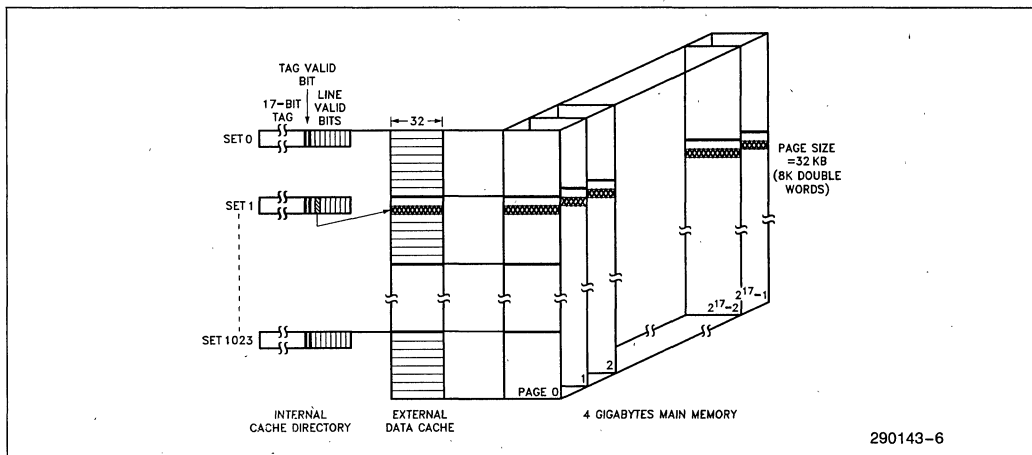


Figure 2-1. Direct Mapped Cache Organization

Each block in the external cache has an associated 26-bit entry in the 82385's internal cache directory. This entry has three components: a 17-bit "tag," a "tag valid" bit, and eight "line valid" bits. The tag acts as a main memory page number (17 tag bits support 2^{17} pages). For example, if line 9 of page 2 currently resides in the cache, then a binary 2 is stored in the Set 1 tag field. (For any 82385 direct mapped cache page in main memory, Set 0 consists of lines 0–7, Set 1 consists of lines 8–15, etc. Line 9 is shaded in Figure 2-1.) An important characteristic of a direct mapped cache is that line 9 of any page can only reside in line 9 of the cache. All identical page offsets map to a single cache location.

The data in a cache set is considered valid or invalid depending on the status of its tag valid bit. If clear, the entire set is considered invalid. If true, an individual line within the set is considered valid or invalid depending on the status of its line valid bit.

The 82385 sees the 386 DX address bus (A2–A31) as partitioned into three fields: a 17-bit "tag" field (A15–A31), a 10-bit "set-address" field (A5–A14), and a 3-bit "line select" field (A2–A4). (See Figure 2-2.) The lower 13 address bits (A2–A14) also serve as the "cache address" which directly selects one of 8K doublewords in the external cache.

2.1.2 Direct Mapped Cache Operation

The following is a description of the interaction between the 386 DX, cache, and cache directory.

2.1.2.1 READ HITS

When the 386 DX initiates a memory read cycle, the 82385 uses the 10-bit set address to select one of

1024 directory entries, and the 3-bit line select field to select one of eight line valid bits within the entry. The 13-bit cache address selects the corresponding doubleword in the cache. The 82385 compares the 17-bit tag field (A15–A31 of the 386 DX access) with the tag stored in the selected directory entry. If the tag and upper address bits match, and if both the tag and upper address bits are set, the result is a hit, and the 82385 directs the cache to drive the selected doubleword onto the 386 DX data bus. A read hit does not alter the contents of the cache or directory.

2.1.2.2 READ MISSES

A read miss can occur in two ways. The first is known as a "line" miss, and occurs when the tag and upper address bits match and the tag valid bit is set, but the line valid bit is clear. The second is called a "tag" miss, and occurs when either the tag and upper address bits do not match, or the tag valid bit is clear. (The line valid bit is a "don't care" in a tag miss.) In both cases, the 82385 forwards the 386 DX reference to the system, and as the returning data is fed to the 386 DX, it is written into the cache and validated in the cache directory.

In a line miss, the incoming data is validated simply by setting the previously clear line valid bit. In a tag miss, the upper address bits overwrite the previously stored tag, the tag valid bit is set, the appropriate line valid bit is set, and the other seven line valid bits are cleared. Subsequent tag hits with line misses will only set the appropriate line valid bit. (Any data associated with the previous tag is no longer considered resident in the cache.)

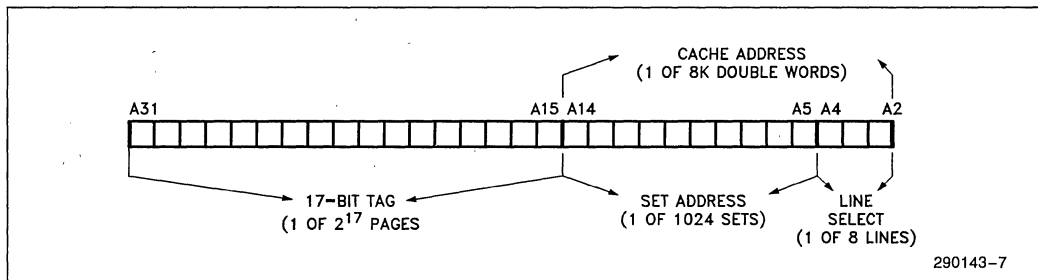


Figure 2-2. 386 DX Address Bus Bit Fields—Direct Mapped Organization

2.1.2.3 OTHER OPERATIONS THAT AFFECT THE CACHE AND CACHE DIRECTORY

The other operations that affect the cache and/or directory are write hits, snoop hits, cache flushes, and 82385 resets. In a write hit, the cache is updated along with main memory, but the directory is unaffected. In a snoop hit, the cache is unaffected, but the affected line is invalidated by clearing its line valid bit in the directory. Both an 82385 reset and cache flush clear all tag valid bits.

When an 386 DX CPU/82385 system "wakes up" upon reset, all tag valid bits are clear. At this point, a read miss is the only mechanism by which main memory data is copied into the cache and validated in the cache directory. Assume an early 386 DX code access seeks (for the first time) line 9 of page 2. Since the tag valid bit is clear, the access is a tag miss, and the data is fetched from main memory. Upon return, the data is fed to the 386 DX and simultaneously written into line 9 of the cache. The set directory entry is updated to show this line as valid. Specifically, the tag and appropriate line valid bits are set, the remaining seven line valid bits cleared, and a binary 2 written into the tag. Since code is sequential in nature, the 386 DX will likely next want line 10 of page 2, then line 11, and so on. If the 386 DX sequentially fetches the next six lines, these fetches will be line misses, and as each is fetched from main memory and written into the cache, its corresponding line valid bit is set. This is the basic

flow of events that fills the cache with valid data. Only after a piece of data has been copied into the cache and validated can it be accessed in a zero wait state read hit. Also, a cache entry must have been validated before it can be subsequently altered by a write hit, or invalidated by a snoop hit.

An extreme example of "thrashing" is if line 9 of page two is an instruction to jump to line 9 of page one, which is an instruction to jump back to line 9 of page two. Thrashing results from the direct mapped cache characteristic that all identical page offsets map to a single cache location. In this example, the page one access overwrites the cached page two data, and the page two access overwrites the cached page one data. As long as the code jumps back and forth the hit rate is zero. This is of course an extreme case. The effect of thrashing is that a direct mapped cache exhibits a slightly reduced overall hit rate as compared to a set associative cache of the same size.

2.2 TWO WAY SET ASSOCIATIVE CACHE

2.2.1 Two Way Set Associative Cache Structure and Terminology

Figure 2-3 illustrates the relationship between the directory, cache, and 4 Gigabyte address space.

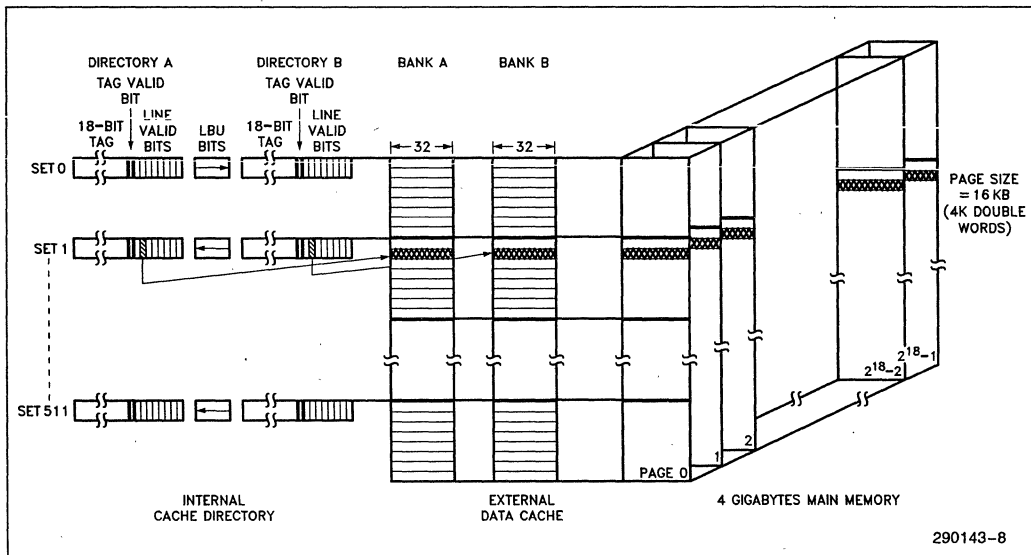


Figure 2-3. Two-Way Set Associative Cache Organization

Whereas the direct mapped cache is organized as one bank of 8K doublewords, the two way set associative cache is organized as two banks (A and B) of 4K doublewords each. The page size is halved, and the number of pages doubled. (Note the extra tag bit.) The cache now has 512 sets in each bank. (Two banks times 512 sets gives a total of 1024. The structure can be thought of as two half-sized direct mapped caches in parallel.) The performance advantage over a direct mapped cache is that all identical page offsets map to two cache locations instead of one, reducing the potential for thrashing. The 82385's partitioning of the 386 DX address bus is depicted in Figure 2-4.

2.2.2 LRU Replacement Algorithm

The two way set associative directory has an additional feature: the "least recently used" or LRU bit. In the event of a read miss, either bank A or bank B will be updated with new data. The LRU bit flags the candidate for replacement. Statistically, of two blocks of data, the block most recently used is the block most likely to be needed again in the near future. By flagging the least recently used block, the 82385 ensures that the cache block replaced is the least likely to have data needed by the CPU.

2.2.3 Two Way Set Associative Cache Operation

2.2.3.1 READ HITS

When the 386 DX initiates a memory read cycle, the 82385 uses the 9-bit set address to select one of 512 sets. The two tags of this set are simultaneously compared with A14-A31, both tag valid bits checked, and both appropriate line valid bits checked. If either comparison produces a hit, the corresponding cache bank is directed to drive the selected doubleword onto the 386 DX data bus. (Note that both banks will never concurrently cache the same main memory location.) If the requested data resides in bank A, the LRU bit is pointed toward

B. If B produces the hit, the LRU bit is pointed toward A.

2.2.3.2 READ MISSES

As in direct mapped operation, a read miss can be either a line or tag miss. Let's start with a tag miss example. Assume the 386 DX seeks line 9 of page 2, and that neither the A or B directory produces a tag match. Assume also, as indicated in Figure 2-3, that the LRU bit points to A. As the data returns from main memory, it is loaded into offset 9 of bank A. Concurrently, this data is validated by updating the set 1 directory entry for bank A. Specifically, the upper address bits overwrite the previous tag, the tag valid bit is set, the appropriate line valid bit is set, and the other seven line valid bits cleared. Since this data is the most recently used, the LRU bit is turned toward B. No change to bank B occurs.

If the next 386 DX request is line 10 of page two, the result will be a line miss. As the data returns from main memory, it will be written into offset 10 of bank A (tag hit/line miss in bank A), and the appropriate line valid bit will be set. A line miss in one bank will cause the LRU bit to point to the other bank. In this example, however, the LRU bit has already been turned toward B.

2.2.3.3 OTHER OPERATIONS THAT AFFECT THE CACHE AND CACHE DIRECTORY

Other operations that affect the cache and cache directory are write hits, snoop hits, cache flushes, and 82385 resets. A write hit updates the cache along with main memory. If directory A detects the hit, bank A is updated. If directory B detects the hit, bank B is updated. If one bank is updated, the LRU bit is pointed toward the other.

If a snoop hit invalidates an entry, for example, in cache bank A, the corresponding LRU bit is pointed toward A. This ensures that invalid data is the prime candidate for replacement in a read miss. Finally, resets and flushes behave just as they do in a direct mapped cache, clearing all tag valid bits.

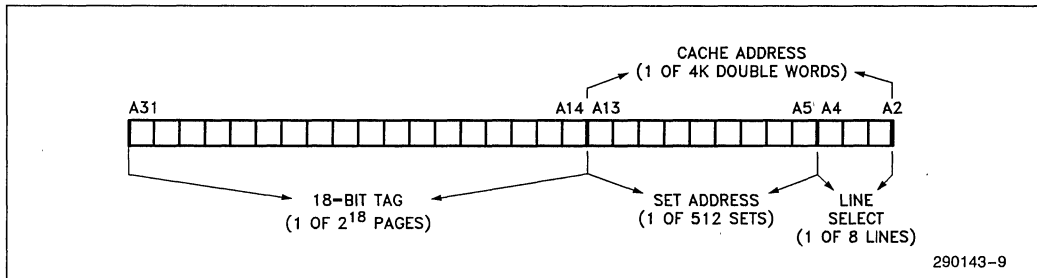


Figure 2-4. 386 DX Address Bus Bit Fields—Two-Way Set Associative Organization

3.0 82385 PIN DESCRIPTION

The 82385 creates the 82385 local bus, which is a functional 386 DX interface. To facilitate understanding, 82385 local bus signals go by the same name as their 386 DX equivalents, except that they are preceded by the letter "B". The 82385 local bus equivalent to ADS# is BADS#, the equivalent to NA# is BNA#, etc. This convention applies to bus states as well. For example, BT1P is the 82385 local bus state equivalent to the 386 DX T1P state.

3.1 386 DX CPU/82385 INTERFACE SIGNALS

These signals form the direct interface between the 386 DX and 82385.

3.1.1 386 DX CPU/82385 Clock (CLK2)

CLK2 provides the fundamental timing for an 386 DX CPU/82385 system, and is driven by the same source that drives the 386 DX CLK2 input. The 82385, like the 386 DX, divides CLK2 by two to generate an internal "phase indication" clock. (See Figure 3-1.) The CLK2 period whose rising edge drives the internal clock low is called PHI1, and the CLK2 period that drives the internal clock high is called PHI2. A PHI1-PHI2 combination (in that order) is

known as a "T" state, and is the basis for 386 DX bus cycles.

3.1.2 386 DX CPU/82385 Reset (RESET)

This input resets the 82385, bringing it to an initial known state, and is driven by the same source that drives the 386 DX RESET input. A reset effectively flushes the cache by clearing all cache directory tag valid bits. The falling edge of RESET is synchronized to CLK2, and used by the 82385 to properly establish the phase of its internal clock. (See Figure 3-2.) Specifically, the second internal phase following the falling edge of RESET is PHI2.

3.1.3 386 DX CPU/82385 Address Bus (A2-A31), Byte Enables (BE0# - BE3#), and Cycle Definition Signals (M/IO#, D/C#, W/R#, LOCK#)

The 82385 directly connects to these 386 DX outputs. The 386 DX address bus is used in the cache directory comparison to see if data referenced by 386 DX resides in the cache, and the byte enables inform the 82385 as to which portions of the data bus are involved in an 386 DX cycle. The cycle definition signals are decoded by the 82385 to determine the type of cycle the 386 DX is executing.

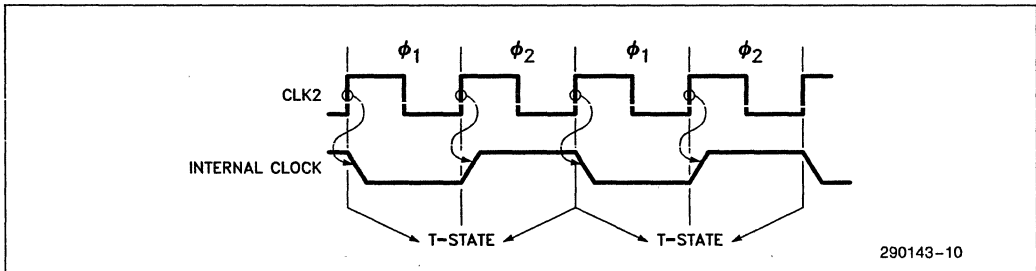


Figure 3-1. CLK2 and Internal Clock

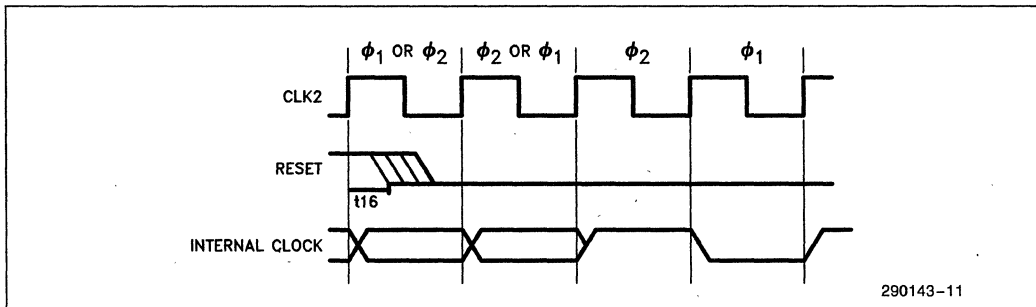


Figure 3-2. Reset/Internal Phase Relationship

3.1.4 386 DX CPU/82385 Address Status (ADS#) and Ready Input (READYI#)

ADS#, a 386 DX output, tells the 82385 that new address and cycle definition information is available. READYI#, an input to both the 386 DX (via the 386 DX READY# input pin) and 82385, indicates the completion of a 386 DX bus cycle. ADS# and READYI# are used to keep track of the 386 DX bus state.

3.1.5 386 DX Next Address Request (NA#)

This 82385 output controls 386 DX pipelining. It can be tied directly to the 386 DX NA# input, or it can be logically "AND"ed with other 386 DX local bus next address requests.

3.1.6 Ready Output (READYO#) and Bus Ready Enable (BRDYEN#)

The 82385 directly terminates all but two types of 386 DX bus cycles with its READYO# output. 386 DX local bus cycles must be terminated by the local device being accessed. This includes devices decoded using the 82385 LBA# signal and 80387 accesses. The other cycles not directly terminated by the 82385 are 82385 local bus reads, specifically cache read misses and non-cacheable reads. (Recall that the 82385 forwards and runs such cycles on the 82385 bus.) In these cycles the signal that terminates the 82385 local bus access is BREADY#, which is gated through to the 386 DX local bus such that the 386 DX and 82385 local bus cycles are concurrently terminated. BRDYEN# is used to gate the BREADY# signal to the 386 DX.

3.2 CACHE CONTROL SIGNALS

These 82385 outputs control the external 32 KB cache data memory.

3.2.1 Cache Address Latch Enable (CALEN)

This signal controls the latch (typically an F or AS series 74373) that resides between the low order 386 DX address bits and the cache SRAM address inputs. (The outputs of this latch are the "cache address" described in the previous chapter.) When CALEN is high the latch is transparent. The falling edge of CALEN latches the current inputs which remain applied to the cache data memory until CALEN returns to an active high state.

3.2.2 Cache Transmit/Receive (CT/R#)

This signal defines the direction of an optional data transceiver (typically an F or AS series 74245) between the cache and 386 DX data bus. When high, the transceiver is pointed towards the 386 DX local data bus (the SRAMs are output enabled). When low, the transceiver points towards the cache data memory. A transceiver is required if the cache is designed with SRAMs that lack an output enable control. A transceiver may also be desirable in a system that has a heavily loaded 386 DX local data bus. These devices are not necessary when using SRAMs which incorporate an output enable.

3.2.3 Cache Chip Selects (CS0# - CS3#)

These active low signals tie to the cache SRAM chip selects, and individually enable the four bytes of the 32-bit wide cache. CS0# enables D0-D7, CS1# enables D8-D15, CS2# enables D16-D23, and CS3# enables D24-D31. During read hits, all four bytes are enabled regardless of whether or not all four 386 DX byte enables are active. (The 386 DX ignores what it did not request.) Also, all four cache bytes are enabled in a read miss so as to update the cache with a complete line (double word). In a write hit, only those cache bytes that correspond to active byte enables are selected. This prevents cache data from being corrupted in a partial doubleword write.

3.2.4 Cache Output Enables (COEA#, COEB#) and Write Enables (CWEA#, CWEB#)

COEA# and COEB# are active low signals which tie to the cache SRAM or Transceiver output enables and respectively enable cache bank A or B. The state of DEFOE# (define cache output enable), an 82385 configuration input, determines the functional definition of COEA# and COEB#.

If DEFOE# = V_{IL}, in a two-way set associative cache, either COEA# or COEB# is active during read hit cycles only, depending on which bank is selected. In a direct mapped cache, both are activated during read hits, so the designer is free to use either one. This COEx# definition best suites cache SRAMs with output enables.

If DEFOE# = V_{IH}, COEx# is active during read hit, read miss (cache update) and write hit cycles only. This COEx# definition suites cache SRAMs without output enables. In such systems, transceivers are needed and their output enables must be active for writing, as well as reading, the cache SRAMs.

CWEA# and CWEB# are active low signals which tie to the cache SRAM write enables, and respectively enable cache bank A or B to receive data from the 386 DX data bus (386 DX write hit or read miss update). In a two-way set associative cache, one or the other is enabled in a read miss or write hit. In a direct mapped cache, both are activated, so the designer is free to use either one.

The various cache configurations supported by the 82385 are described in Chapter 4.

3.3 386 DX LOCAL BUS DECODE INPUTS

These 82385 inputs are generated by decoding the 386 DX address and cycle definition lines. These active low inputs are sampled at the end of the first state in which the address of a new 386 DX cycle becomes available (T1 or first T2P).

3.3.1 386 DX Local Bus Access (LBA#)

This input identifies an 386 DX access as directed to a resource (other than the cache) on the 386 DX local bus. (The 387 Numerics Coprocessor is considered a 386 DX local bus resource, but LBA# need not be generated as the 82385 internally decodes 387 accesses.) The 82385 simply ignores these cycles. They are neither forwarded to the system nor do they affect the cache or cache directory. Note that LBA# has priority over all other types of cycles. If LBA# is asserted, the cycle is interpreted as an 386 DX local bus access, regardless of the cycle type or status of NCA# or X16#. This allows any 386 DX cycle (memory, I/O, interrupt acknowledge, etc.) to be kept on the 386 local bus if desired.

3.3.2 Non-Cacheable Access (NCA#)

This active low input identifies a 386 DX cycle as non-cacheable. The 82385 forwards non-cacheable cycles to the 82385 local bus and runs them. The cache and cache directory are unaffected.

NCA# allows a designer to set aside a portion of main memory as non-cacheable. Potential applications include memory-mapped I/O and systems where multiple masters access dual ported memory via different busses. Another possibility makes use of the 386 DX D/C# output. The 82385 by default implements a unified code and data cache, but driving NCA# directly by D/C# creates a data only cache. If D/C# is inverted first, the result is a code only cache.

3.3.3 16-Bit Access (X16#)

X16# is an active low input which identifies 16-bit memory and/or I/O space, and the decoded signal that drives X16# should also drive the 386 DX BS16# input. 16-bit accesses are treated like non-cacheable accesses: they are forwarded to and executed on the 82385 local bus with no impact on the cache or cache directory. In addition, the 82385 locks the two halves of a non-aligned 16-bit transfer from interruption by another master, as does the 386 DX.

3.4 82385 LOCAL BUS INTERFACE SIGNALS

The 82385 presents a "386 DX-like" front end to the system, and the signals discussed in this section are 82385 local bus equivalents to actual 386 DX signals. These signals are named with respect to their 386 DX counterparts, but with the letter "B" appended to the front.

Note that the 82385 itself does not have equivalent output signals to the 386 DX data bus (D0–D31), address bus (A2–A31), and cycle definition signals (M/IO#, D/C#, W/R#). The 82385 data bus (BD0–BD31) is actually the system side of a latching transceiver, and the 82385 address bus and cycle definition signals (BA2–BA31, BM/IO#, BD/C#, BW/R#) are the outputs of an edge-triggered latch. The signals that control this data transceiver and address latch are discussed in Section 3.5.

3.4.1 82385 Bus Byte Enables (BBE0#–BBE3#)

BBE0#–BBE3# are the 82385 local bus equivalents to the 386 DX byte enables. In a cache read miss, the 82385 drives all four signals low, regardless of whether or not all four 386 DX byte enables are active. This ensures that a complete line (doubleword) is fetched from main memory for the cache update. In all other 82385 local bus cycles, the 82385 duplicates the logic levels of the 386 DX byte enables. The 82385 tri-states these outputs when it is not the current bus master.

3.4.2 82385 Bus Lock (BLOCK#)

BLOCK# is the 82385 local bus equivalent to the 386 DX LOCK# output, and distinguishes between locked and unlocked cycles. When the 386 DX runs a locked sequence of cycles (and LBA# is negated), the 82385 forwards and runs the sequence on the 82385 local bus, regardless of whether any locations

referenced in the sequence reside in the cache. A read hit will be run as if it is a read miss, but a write hit will update the cache as well as being completed to system memory. In keeping with 386 DX behavior, the 82385 does not allow another master to interrupt the sequence. BLOCK# is tri-stated when the 82385 is not the current bus master.

3.4.3 82385 Bus Address Status (BADS#)

BADS# is the 82385 local bus equivalent of ADS#, and indicates that a valid address (BA2–BA31, BBEO#–BBE3#) and cycle definition (BM/IO#, BW/R#, BD/C#) is available. It is asserted in BT1 and BT2P states, and is tri-stated when the 82385 does not own the bus.

3.4.4 82385 Bus Ready Input (BREADY#)

82385 local bus cycles are terminated by BREADY#, just as 386 DX cycles are terminated by the 386 DX READY# input. In 82385 local bus read cycles, BREADY# is gated by BRDYEN# onto the 386 DX local bus, such that it terminates both the 386 DX and 82385 local bus cycles.

3.4.5 82385 Bus Next Address Request (BNA#)

BNA# is the 82385 local bus equivalent to the 386 DX NA# input, and indicates that the system is prepared to accept a pipelined address and cycle definition. If BNA# is asserted and the new cycle information is available, the 82385 begins a pipelined cycle on the 82385 local bus.

3.5 82385 BUS DATA TRANSCEIVER AND ADDRESS LATCH CONTROL SIGNALS

The 82385 data bus is the system side of a latching transceiver (typically an F or AS series 74646), and the 82385 address bus and cycle definition signals are the outputs of an edge-triggered latch (F or AS series 74374). The following is a discussion of the 82385 outputs that control these devices. An important characteristic of these signals and the devices they control is that they ensure that BD0–BD31, BA2–BA31, BM/IO#, BD/C#, and BW/R# reproduce the functionality and timing behavior of their 386 DX equivalents.

3.5.1 Local Data Strobe (LDSTB), Data Output Enable (DOE#), and Bus Transmit/Receive (BT/R#)

These signals control the latching data transceiver. BT/R# defines the transceiver direction. When high, the transceiver drives the 82385 data bus in write cycles. When low, the transceiver drives the 386 DX data bus in 82385 local bus read cycles. DOE# enables the transceiver outputs.

The rising edge of LDSTB latches the 386 DX data bus in all write cycles. The interaction of this signal and the latching transceiver is used to perform the 82385's posted write capability.

3.5.2 Bus Address Clock Pulse (BACP) and Bus Address Output Enable (BAOE#)

These signals control the latch that drives BA2–BA31, BM/IO#, BW/R#, and BD/C#. In any 386 DX cycle that is forwarded to the 82385 local bus, the rising edge of BACP latches the 386 DX address and cycle definition signals. BAOE# enables the latch outputs when the 82385 is the current bus master and disables them otherwise.

3.6 STATUS AND CONTROL SIGNALS

3.6.1 Cache Miss Indication (MISS#)

This output accompanies cacheable read and write miss cycles. This signal transitions to its active low state when the 82385 determines that a cacheable 386 DX access is a miss. Its timing behavior follows that of the 82385 local bus cycle definition signals (BM/IO#, BD/C#, BW/R#) so that it becomes available with BADS# in BT1 or the first BT2P. MISS# is floated when the 82385 does not own the bus, such that multiple 82385's can share the same node in multi-cache systems. (As discussed in Chapter 7, this signal also serves a reserved function in testing the 82385.)

3.6.2 Write Buffer Status (WBS)

The latching data transceiver is also known as the "posted write buffer." WBS indicates that this buffer contains data that has not yet been written to the system even though the 386 DX may have begun its next cycle. It is activated when 386 DX data is latched, and deactivated when the corresponding

82385 local bus write cycle is completed (BREADY#). (As discussed in Chapter 7, this signal also serves a reserved function in testing the 82385.)

WBS can serve several functions. In multi-processor applications, it can act as a coherency mechanism by informing a bus arbiter that it should let a write cycle run on the system bus so that main memory has the latest data. If any other 82385 cache subsystems are on the bus, they will monitor the cycle via their bus watching mechanisms. Any 82385 that detects a snoop hit will invalidate the corresponding entry in its local cache.

3.6.3 Cache Flush (FLUSH)

When activated, this signal causes the 82385 to clear all of its directory tag valid bits, effectively flushing the cache. (As discussed in Chapter 7, this signal also serves a reserved function in testing the 82385.) The primary use of the FLUSH input is for diagnostics and multi-processor support. The use of this pin as a coherency mechanism may impact software transparency.

The FLUSH input must be held active for at least 4 CLK (8 CLK₂) cycles to complete the flush sequence. If FLUSH is still active after 4 CLK cycles, any accesses to the cache will be misses and the cache will not be updated (since FLUSH is active).

3.7 BUS ARBITRATION SIGNALS (BHOLD AND BHLDA)

In master mode, BHOLD is an input that indicates a request by a slave device for bus ownership. The 82385 acknowledges this request via its BHLDA output. (These signals function identically to the 386 DX HOLD and HLDA signals.)

The roles of BHOLD and BHLDA are reversed for an 82385 in slave mode. BHOLD is now an output indicating a request for bus ownership, and BHLDA an input indicating that the request has been granted.

3.8 COHERENCY (BUS WATCHING) SUPPORT SIGNALS (SA2-SA31, SSTB#, SEN)

These signals form the 82385's bus watching interface. The Snoop Address Bus (SA2-SA31) connects to the system address lines if masters reside at both the system and 82385 local bus levels, or the 82385 local bus address lines if masters reside only at the 82385 local bus level. Snoop Strobe (SSTB#) indicates that a valid address is on the

snoop address inputs. Snoop Enable (SEN) indicates that the cycle is a write. In a system with masters only at the 82385 local bus level, SA2-SA31, SSTB#, and SEN can be driven respectively by BA2-BA31, BADS#, and BW/R# without any support circuitry.

3.9 CONFIGURATION INPUTS (2W/D#, M/S#, DEFOE#)

These signals select the configurations supported by the 82385. They are hardware strap options and must not be changed dynamically. 2W/D# (2-Way/Direct Mapped Select) selects a two-way set associative cache when tied high, or a direct mapped cache when tied low. M/S# (Master/Slave Select) chooses between master mode (M/S# high) and slave mode (M/S# low). DEFOE# defines the functionality of the 82385 cache output enables (COEA# and COEB#). DEFOE# allows the 82385 to interface to SRAMs with output enables (DEFOE# low) or to SRAMs requiring transceivers (DEFOE# high).

4.0 386 DX LOCAL BUS INTERFACE

The following is a detailed description of how the 82385 interfaces to the 386 DX and to 386 DX local bus resources. Items specifically addressed are the interfaces to the 386 DX, the cache SRAMs, and the 387 Numerics Coprocessor.

The many timing diagrams in this and the next chapter provide insight into the dual pipelined bus structure of a 386 DX CPU/82385 system. It's important to realize, however, that one need not know every possible cycle combination to use the 82385. The interface is simple, and the dual bus operation invisible to the 386 DX and system. To facilitate discussion of the timing diagrams, several conventions have been adopted. Refer to Figure 4-2A, and note that 386 DX bus cycles, 386 DX bus states, and 82385 bus states are identified along the top. All states can be identified by the "frame numbers" along the bottom. The cycles in Figure 4-2A include a cache read hit (CRDH), a cache read miss (CRDM), and a write (WT). WT represents any write, cacheable or not. When necessary to distinguish cacheable writes, a write hit goes by CWT_H and a write miss by CWT_M. Non-cacheable system reads go by SBRD. Also, it is assumed that system bus pipelining occurs even though the BNA# signal is not shown. When the system pipeline begins is a function of the system bus controller.

386 DX bus cycles can be tracked by ADS# and READYI#, and 82385 cycles by BADS# and BREADY#. These four signals are thus a natural

choice to help track parallel bus activity. Note in the timing diagrams that 386 DX cycles are numbered using ADS# and READYI#, and 82385 cycles using BADS# and BREADY#. For example, when the address of the first 386 DX cycle becomes available, the corresponding assertion of ADS# is marked "1", and the READYI# pulse that terminates the cycle is marked "1" as well. Whenever a 386 DX cycle is forwarded to the system, its number is forwarded as well so that the corresponding 82385 bus cycle can be tracked by BADS# and BREADY#.

The "N" value in the timing diagrams is the assumed number of main memory wait states inserted in a non-pipelined 82385 bus cycle. For example, a non-pipelined access to N=2 memory requires a total of four bus states, while a pipelined access requires three. (The pipeline advantage effectively hides one main memory wait state.)

4.1 PROCESSOR INTERFACE

This section presents the 386 DX CPU /82385 hardware interface and discusses the interaction and timing of this interface. Also addressed is how to decode the 386 DX address bus to generate the

82385 inputs LBA#, NCA#, and X16#. (Recall that LBA# allows memory and/or I/O space to be set aside for 386 DX local bus resources; NCA# allows system memory to be set aside as non-cacheable; and X16# allows system memory and/or I/O space to be reserved for 16-bit resources.) Finally, the 82385's handling of 16-bit space is discussed.

4.1.1 Hardware Interface

Figure 4-1 is a diagram of an 386 DX CPU/82385 system, which can be thought of as three distinct interfaces. The first is the 386 DX CPU/82385 interface (including the Ready Logic). The second is the cache interface, as depicted by the cache control bus in the upper left corner of Figure 4-1. The third is the 82385 bus interface, which includes both direct connects and signals that control the 74374 address/cycle definition latch and 74646 latching data transceiver. (The 82385 bus interface is the subject of the next chapter.)

As seen in Figure 4-1, the 386 DX CPU/82385 interface is a straightforward connection. The only necessary support logic is that required to sum all ready sources.

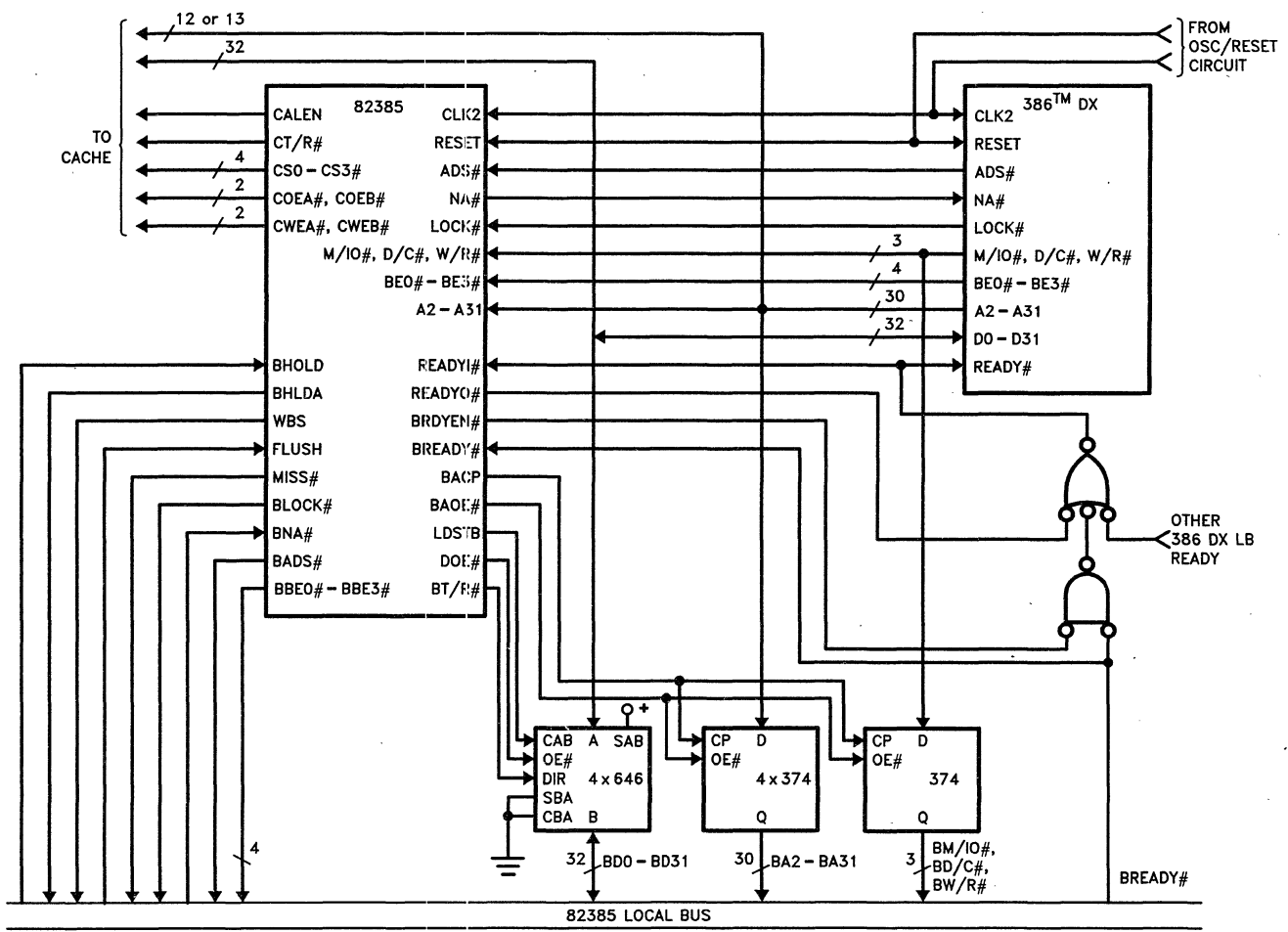


Figure 4-1. 386 DX CPU/82385 Interface

4-360

4.1.2 Ready Generation

Note in Figure 4-1 that the ready logic consists of two gates. The upper three-input AND gate (shown as a negative logic OR) sums all 386 DX local bus ready sources. One such source is the 82385 READYO# output, which terminates read hits and posted writes. The output of this gate drives the 386 DX READY# input and is monitored by the 82385 (via READYI#) to track the 386 DX bus state.

When the 82385 forwards a 386 DX read cycle to the 82385 bus (cache read miss or non-cacheable read), it does not directly terminate the cycle via READYO#. Instead, the 386 DX and 82385 bus cycles are concurrently terminated by a system ready

source. This is the purpose of the additional two-input OR gate (negative logic AND) in Figure 4-1. When the 82385 forwards a read to the 82385 bus, it asserts BRDYEN# which enables the system ready signal (BREADY#) to directly terminate the 386 DX bus cycle.

Figures 4-2A and 4-2B illustrate the behavior of the signals involved in ready generation. Note in cycle 1 of Figure 4-2A that the 82385 READYO# directly terminates the hit cycle. In cycle 2, READYO# is not activated. Instead the 82385 BRDYEN# is activated in BT2, BT2P, or BT2I states such that BREADY# can concurrently terminate the 386 DX and 82385 bus cycles (frame 6). Cycle 3 is a posted write. The write data becomes available in T1P (frame 7), and

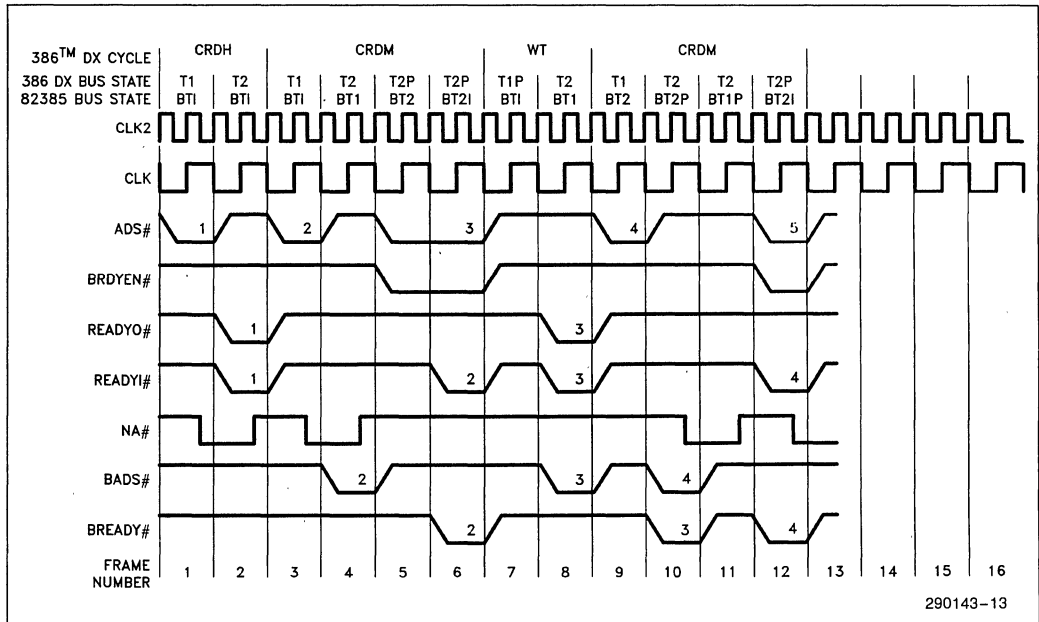


Figure 4-2A. READYO#, BRDYEN#, and NA# (N = 1)

290143-13

the address, data, and cycle definition of the write are latched in T2 (frame 8). The 386 DX cycle is terminated by **READYO#** in frame 8 with no wait states. The 82385, however, sees the write cycle through to completion on the 82385 bus where it is terminated in frame 10 by **BREADY#**. In this case, the **BREADY#** signal is not gated through to the 386 DX. Refer to Figures 4-2A and 4-2B for clarification.

4.1.3. NA# and 386 DX Local Bus Pipelining

Cycle 1 of Figure 4-2A is a typical cache read hit. The 386 DX address becomes available in T1, and the 82385 uses this address to determine if the referenced data resides in the cache. The cache look-up is completed and the cycle qualified as a hit or miss in T1. If the data resides in the cache, the cache is directed to drive the 386 DX data bus, and the 82385 drives its **READYO#** output so the cycle can be terminated at the end of the first T2 with no wait states.

Although cycle 2 starts out like cycle 1, at the end of T1 (frame 3), it is qualified as a miss and forwarded to the 82385 bus. The 82385 bus cycle begins one state after the 386 DX bus cycle, implying a one wait state overhead associated with cycle 2 due to the look-up. When the 82385 encounters the miss, it immediately asserts **NA#**, which puts the 386 DX into pipelined mode. Once in pipelined mode, the 82385 is able to qualify an 386 DX cycle using the 386 DX pipelined address and control signals. The result is that the cache look-up state is hidden in all but the first of a contiguous sequence of read misses. This is shown in the first two cycles, both read misses, of Figure 4-2B. The CPU sees the look-up state in the first cycle, but not in the second. In fact, the second miss requires a total of only two states, as not only does 386 DX pipelining hide the look-up state, but system pipelining hides one of the main memory wait states. (System level pipelining via **BNA#** is discussed in the next chapter.) Several characteristics of the 82385's pipelining of the 386 DX are as follows:

- The above discussion applies to all system reads, not just cache read misses.

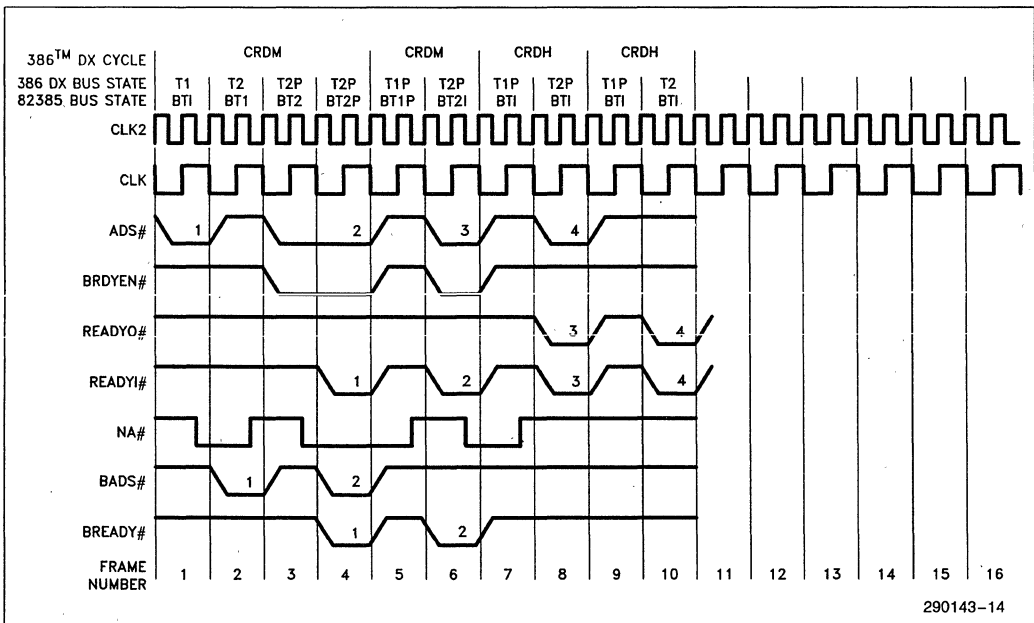


Figure 4-2B. **READYO#**, **BRDYEN#**, and **NA#** (N = 1)

- The 82385 provides the fastest possible switch to pipelining, T1-T2-T2P. The exception to this is when a system read follows a posted write. In this case, the sequence is T1-T2-T2-T2P. (Refer to cycle 4 of Figure 4-2A.) The number of T2 states is dependent on the number of main memory wait states.
- Refer to the read hit in Figure 4-2A (cycle 1), and note that NA# is actually asserted before the end of T1, before the hit/miss decision is made. This is of no consequence since even though NA# is sampled active in T2, the activation of READY# in the same T2 renders NA# a “don’t care”. NA# is asserted in this manner to meet 386 DX timing requirements and to ensure the fastest possible switch to pipelined mode.
- All read hits and the majority of writes can be serviced by the 82385 with zero wait states in non-pipelined mode, and the 82385 accordingly attempts to run all such cycles in non-pipelined mode. An exception is seen in the hit cycles (cycles 3 and 4) of Figure 4-2B. The 82385 does not know soon enough that cycle 3 is a hit, and thus sustains the pipeline. The result is that three sequential hits are required before the 386 DX is totally out of pipelined mode. (The three hits look like T1P-T2P, T1P-T2, T1-T2.) Note that this

does not occur if the number of main memory wait states is equal to or greater than two.

As far as the design is concerned, NA# is generally tied directly to the 386 DX NA# input. However, other local NA# sources may be logically “AND”ed with the 82385 NA# output if desired. It is essential, however, that no device other than the 82385 drive the 386 DX NA# input unless that device resides on the 386 DX local bus in space decoded via LBA#. If desired, the 82385 NA# output can be ignored and the 386 DX NA# input tied high. The 386 DX NA# input should never be tied low, which would always keep it active.

4.1.4 LBA#, NCA#, and X16# Generation

The 82385 input signals LBA#, NCA# and X16# are generated by decoding the 386 DX address (A2–A31) and cycle definition (W/R#, D/C#, M/IO#) lines. The 82385 samples them at the end of the first state in which they become available, which is either T1 or the first T2P cycle. The decode configuration and timings are illustrated respectively in Figures 4-3A and 4-3B.

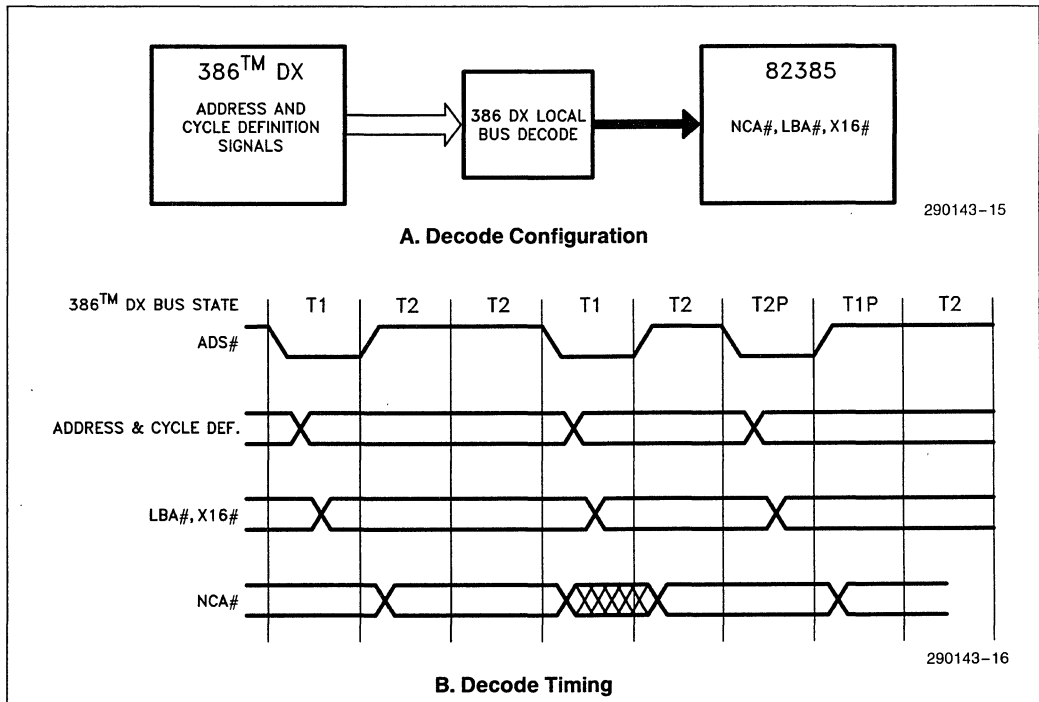


Figure 4-3. NCA#, LBA#, X16# Generation

4.1.5 82385 Handling of 16-Bit Space

As discussed previously, the 82385 does not cache devices decoded as 16-bit. Instead it makes provision to accommodate 16-bit space as non-cacheable via the X16# input. X16# is generated when the user decodes the 386 DX address and cycle definition lines for the BS16# input of the 386 DX (Figure 4-3). The decode output now drives both the 386 DX BS16# input and the 82385 X16# input. Cycles decoded this way are treated as non-cacheable. They are forwarded to and executed on the 82385 bus, but have no impact on the cache or cache directory. The 82385 also monitors the 386 DX byte enables in a 16-bit cycle to see if an additional cycle is required to complete the transfer. Specifically, a second cycle is required if (BE0# OR BE1#) AND (BE2# OR BE3#) is asserted in the current cycle. The 82385, like the 386 DX, will not allow the two halves of a 16-bit transfer to be interrupted by another master. There is an important distinction between the handling of 16-bit space in a 386 DX system with an 82385 as compared to a system without an 82385. The 386 DX BS16# input need not be asserted until the last state of a 16-bit cycle for the 386 DX to recognize it as such. The 82385, however, needs the information earlier, specifically at the end of the first 386 DX bus state (T1 or first T2P) in which the address of the 16-bit cycle becomes available. The result is that in a system without an 82385, 16-bit devices can define themselves as 16-bit devices "on the fly", while in a system with an 82385, 16-bit devices should be located in space set aside for 16-bit devices via the X16# decode.

4.2 CACHE INTERFACE

The following is a description of the external data cache and 82385 cache interface.

4.2.1 Cache Configurations

The 82385 controls the cache memory via the control signals shown in Figure 4-1. These signals drive one of four possible cache configurations, as depicted in Figures 4-4A through 4-4D. Figure 4-4A shows a direct mapped cache organized as 8K doublewords. The likely design choice is four 8K x 8 SRAMs. Figure 4-4B depicts the same cache memory but with a data transceiver between the cache and 386 DX data bus. In this configuration, CT/R# controls the transceiver direction, COEA# drives the transceiver output enable. (COEB# could also be used, and DEFOE# is strapped high.) A data buffer is required if the chosen SRAM does not have a separate output enable. Additionally, buffers may be used to ease SRAM timing requirements or in a system with a heavily loaded data bus. (Guidelines for SRAM selection are included in Chapter 6.)

Figure 4-4C depicts a two-way set associative cache organized as two banks (A and B) of 4K doublewords each. The likely design choice is sixteen 4K x 4 SRAM's. Finally, Figure 4-4D depicts the two-way organization with data buffers between the cache memory and data bus.

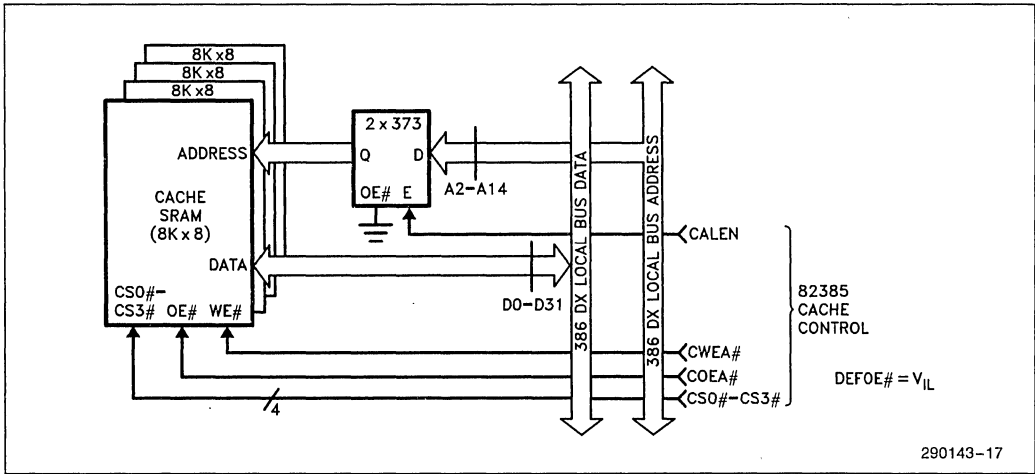


Figure 4-4A. Direct Mapped Cache without Data Buffers

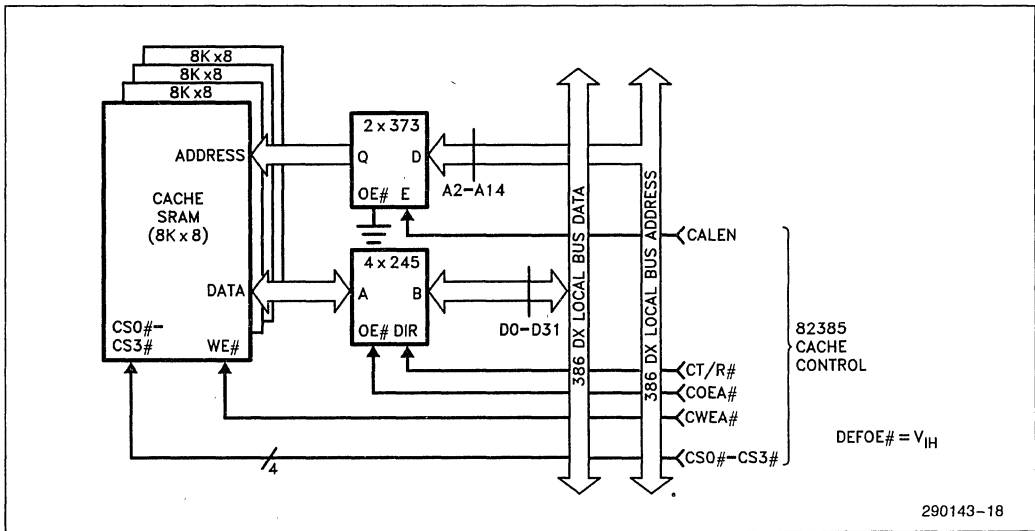


Figure 4-4B. Direct Mapped Cache with Data Buffers

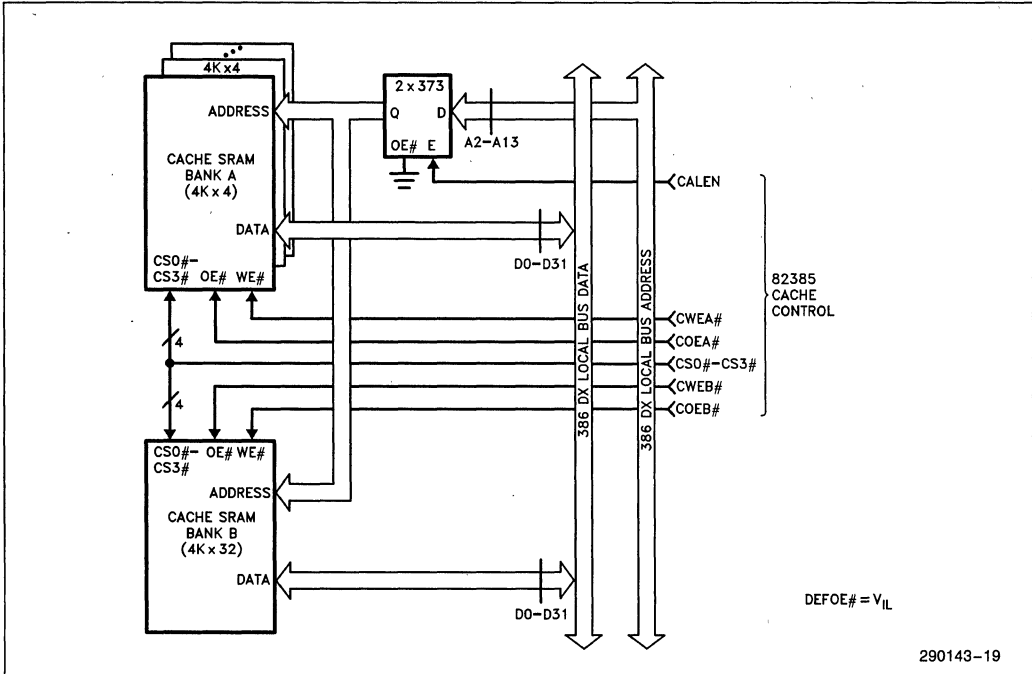


Figure 4-4C. Two-Way Set Associative Cache without Data Buffers

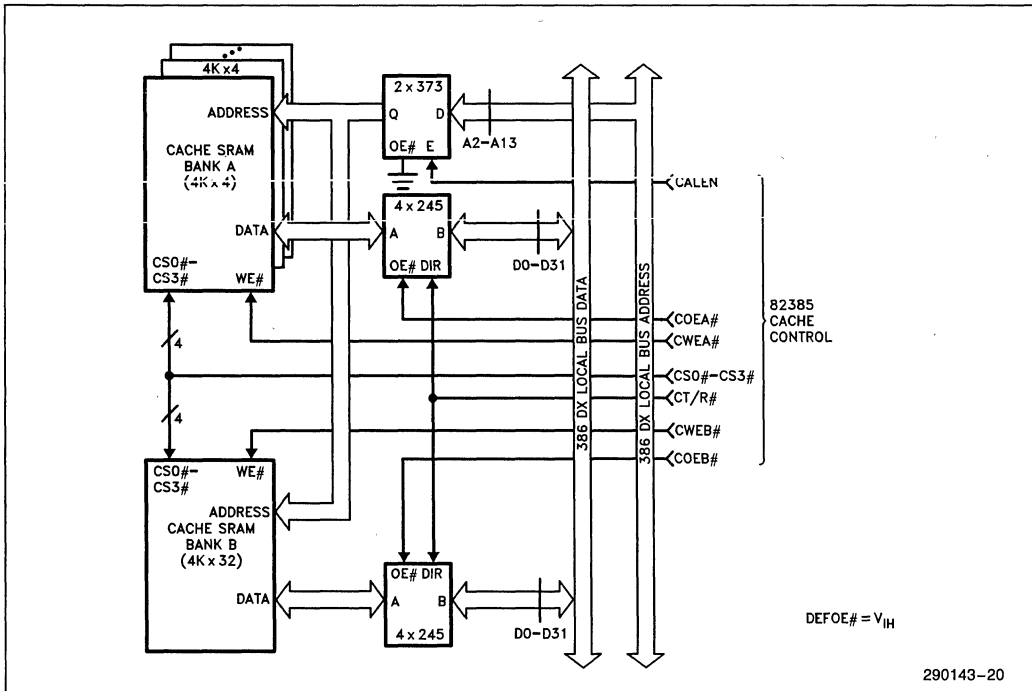


Figure 4-4D. Two-Way Set Associative Cache with Data Buffers

4.2.2 Cache Control—Direct Mapped

Figure 4-5A illustrates the timing of cache read and write hits, while Figure 4-5B illustrates cache updates. In a read hit, the cache output enables are driven from the beginning of T2 (cycle 1 of Figure 4-5A). If at the end of T1 the cycle is qualified as a cacheable read, the output enables are asserted on the assumption that the cycle will be a hit. (Driving the output enables before the actual hit/miss decision is made eases SRAM timing requirements.)

Note that the output enables are asserted at the beginning of T2, but then disabled at the end of T2. Once the output enables are inactive, the 82385 turns the transceiver around (via CT/R#) and drives the write enables to begin the cache update cycle. Note in Figure 4-5B that once the 386 DX is in pipelined mode, the output enables need not be driven prior to a hit/miss decision, since the decision is made earlier via the pipelined address information.

Cycle 1 of Figure 4-5B illustrates what happens when the assumption of a hit turns out to be wrong.

One consequence of driving the output enables low in a miss before the hit/miss decision is made is that since the cache starts driving the 386 DX data bus,

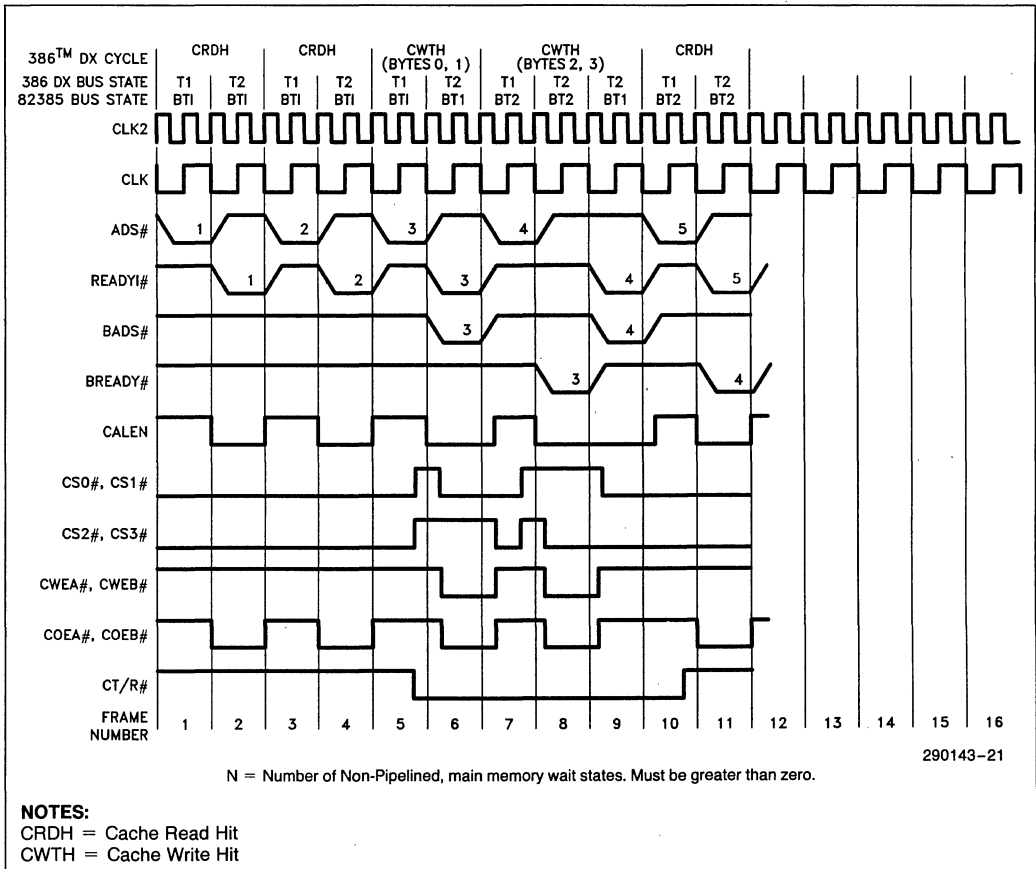


Figure 4-5A. Cache Read and Write Cycles—Direct Mapped (N = 1)

the 82385 cannot enable the 74646 transceiver (Figure 4-1) until after the cache outputs are disabled. (The timing of the 74646 control signals is described in the next chapter.) The result is that the 74646 cannot be enabled soon enough to support N=0 main memory ("N" was defined in section 4.0 as the number of non-pipelined main memory wait states). This means that memory which can run with zero wait states in a non-pipelined cycle should not be mapped into cacheable memory. This should not present a problem, however, as a main memory system built with N=0 memory has no need of a cache. (The main memory is as fast as the cache.) Zero wait state memory can be supported if it is decoded as non-cacheable. The 82385 knows that a cycle is

non-cacheable in time not to drive the cache output enables, and can thus enable the 74646 sooner.

In a write hit, the 82385 only updates the cache bytes that are meant to be updated as directed by the 386 DX byte enables. This prevents corrupting cache data in partial doubleword writes. Note in Figure 4-5A that the appropriate bytes are selected via the cache byte select lines CS0#-CS3#. In a read hit, all four select lines are driven as the 386 DX will simply ignore data it does not need. Also, in a cache update (read miss), all four selects are active in order to update the cache with a complete line (doubleword).

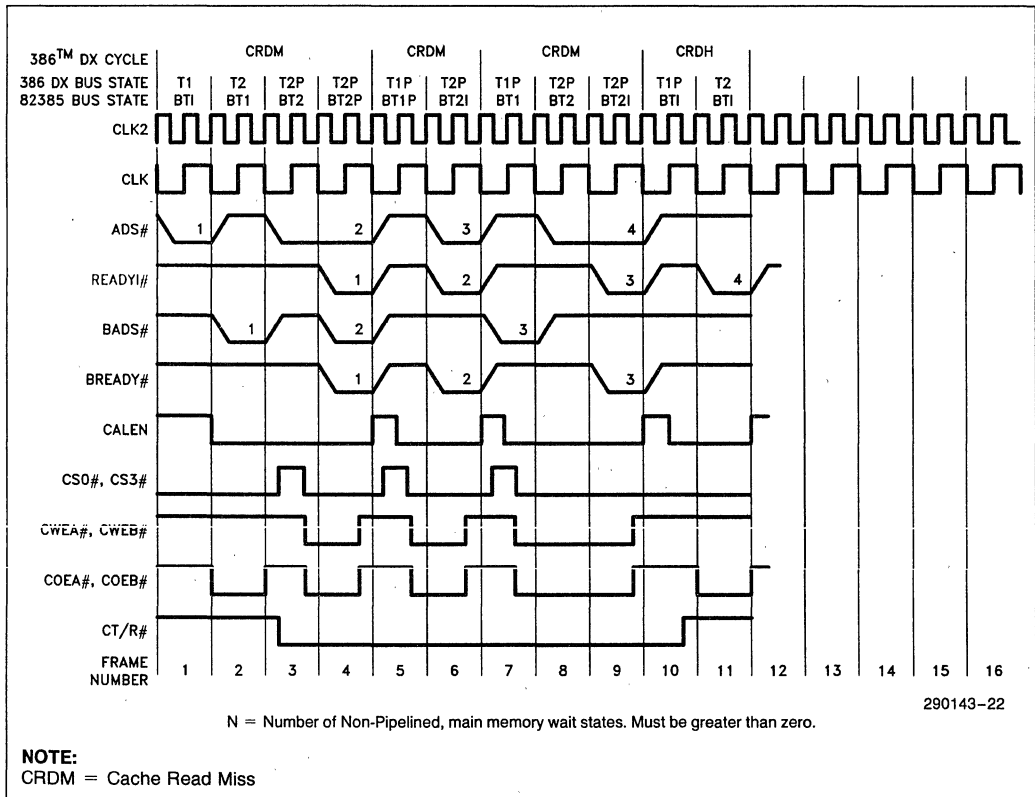


Figure 4-5B. Cache Update Cycles—Direct Mapped (N = 1)

4.2.3 Cache Control—Two-Way Set Associative

Figures 4-6A and 4-6B illustrate the timing of cache read hits, write hits, and updates for a two-way set associative cache. (Note that the cycle sequences are the same as those in Figures 4-5A and 4-5B.) In a cache read hit, only one bank on the other is enabled to drive the 386 DX data bus, so unlike the control of a direct mapped cache, the appropriate cache output enable cannot be driven until the outcome of the hit/miss decision is known. (This implies stricter SRAM timing requirements for a two-way set associative cache.) In write hits and read misses, only one bank or the other is updated.

4.3 387™ DX INTERFACE

The 387 DX Math Coprocessor interfaces to the 386 DX just as it would in a system without an 82385. The 387 DX READY# output is logically "AND"ed along with all other 386 DX local bus ready sources (Figure 4-1), and the output is fed to the 387 DX READY#, 82385 READYI#, and 386 DX READY# inputs.

The 386 DX uniquely addresses the 387 DX by driving M/IO# low and A31 high. The 82385 decodes this internally and treats 387 DX accesses in the same way it treats 386 DX cycles in which LBA# is asserted, it ignores them.

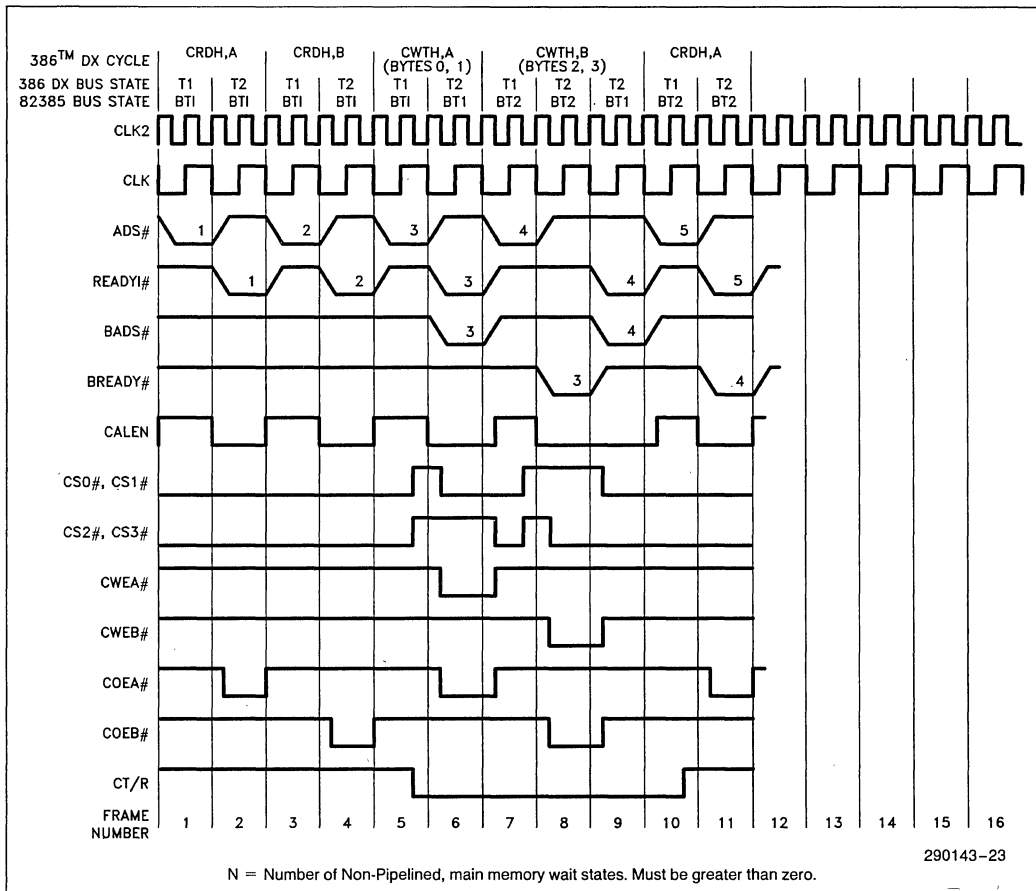


Figure 4-6A. Cache Read and Write Cycles—Two Way Set Associative (N = 1)

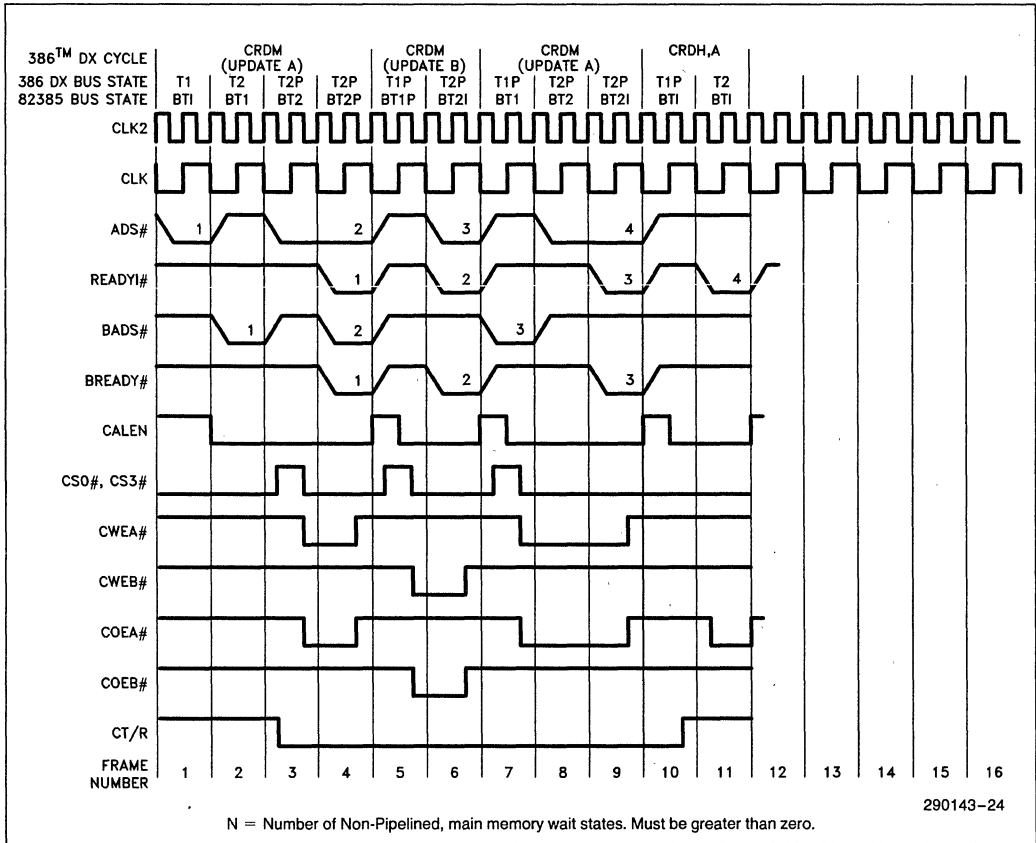


Figure 4-6B. Cache Update Cycles—Two Way Set Associative (N = 1)

5.0 82385 LOCAL BUS AND SYSTEM INTERFACE

The 82385 system interface is the 82385 Local Bus, which presents a "386 DX -like" front end to the system. The system ties to it just as it would to a 386 DX. Although this 386 DX -like front end is functionally equivalent to a 386 DX, there are timing differences which can easily be accounted for in a system design.

The following is a description of the 82385 system interface. After presenting the 82385 bus state machine, the 82385 bus signals are described, as are techniques for accommodating any differences between the 82385 bus and 386 DX bus. Following this is a discussion of the 82385's condition upon reset.

5.1 THE 82385 BUS STATE MACHINE

5.1.1 Master Mode

Figure 5-1A illustrates the 82385 bus state machine when the 82385 is programmed in master mode. Note that it is almost identical to the 386 DX bus state machine, only the bus states are 82385 bus states (BT1P, BTH, etc.) and the state transitions are conditioned by 82385 bus inputs (BNA#, B HOLD, etc.). Whereas a "pending request" to the 386 DX state machine indicates that the 386 DX execution or prefetch unit needs bus access, a pending request to the 82385 state machine indicates that a 386 DX bus cycle needs to be forwarded to the system (read miss, non-cacheable read, write,

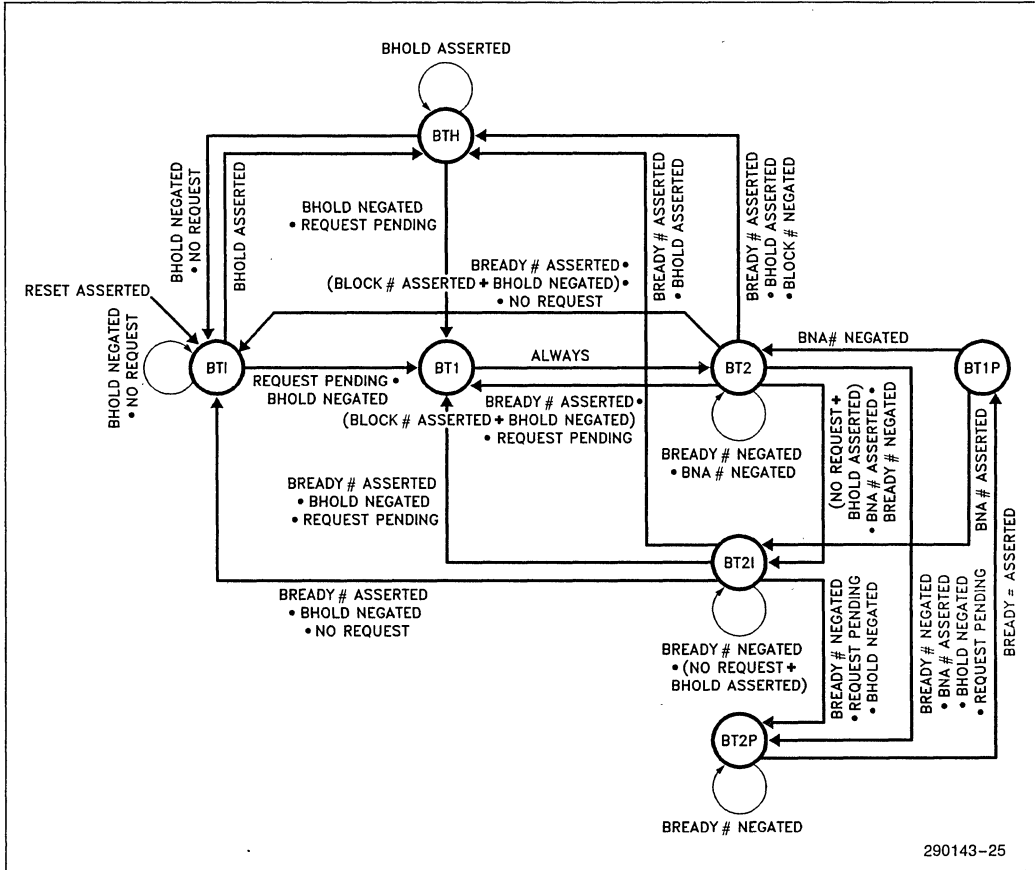


Figure 5-1A. 82385 Local Bus State Machine—Master Mode

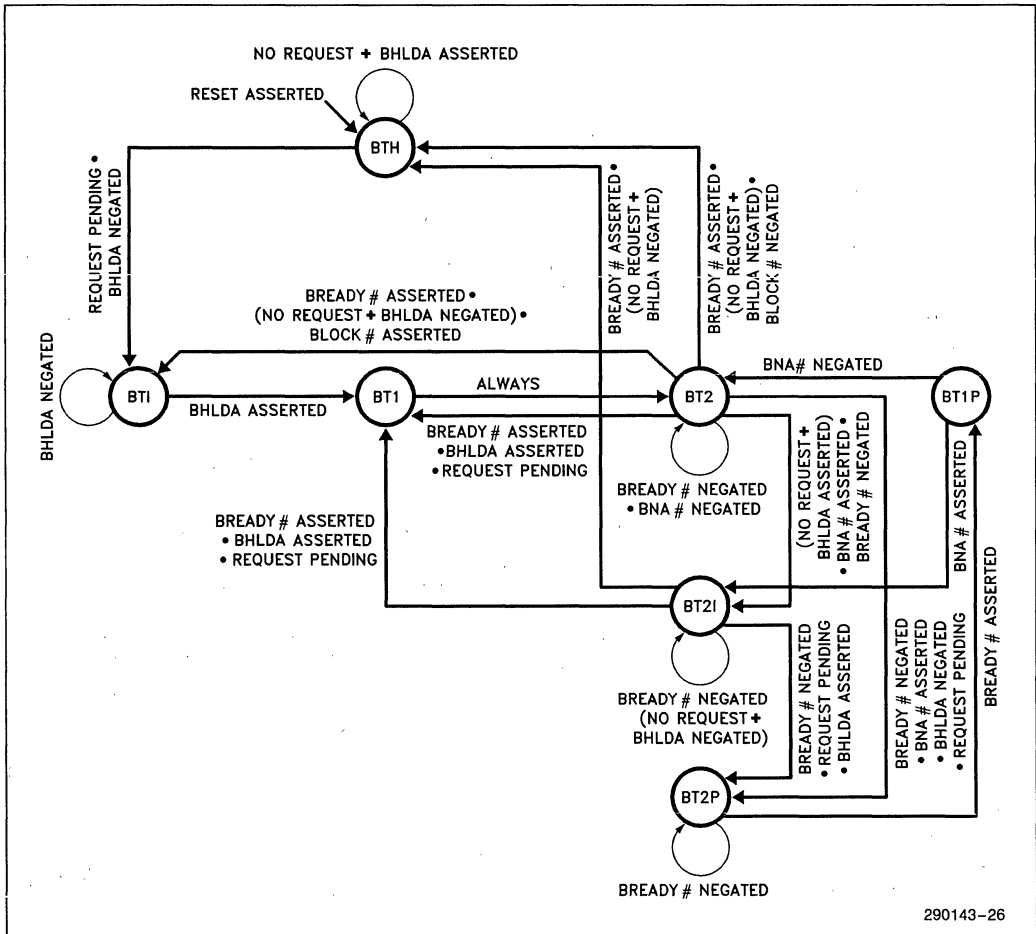


Figure 5-1B. 82385 Local Bus State Machine—Slave Mode

etc.). The only difference between the state machines is that the 82385 does not implement a direct BT1P-BT2P transition. If BNA# is asserted in BT1P, the resulting state sequence is BT1P-BT2I-BT2P. The 82385's ability to sustain a pipeline is not affected by the lack of this state transition.

5.1.2 Slave Mode

The 82385's slave mode state machine (Figure 5-1B) is similar to the master mode machine except that now transitions are conditioned by BHLDA rather than BHOLD. (Recall that in slave mode, the roles of BHOLD and BHLDA are reversed from their master mode roles.) Figure 5-2 clarifies slave mode state machine operation. Upon reset, a slave mode 82385 enters the BTH state. When the 386 DX of the slave 82385 subsystem has a cycle that needs to be forwarded to the system, the 82385 moves to BTI and issues a hold request via BHOLD. It is important to note that a slave mode 82385 does not drive the bus in a BTI state. When the master or bus arbiter returns BHLDA, the slave 82385 enters BT1 and runs

the cycle. When the cycle is completed, and if no additional requests are pending, the 82385 moves back to BTH and disables BHOLD.

If, while a slave 82385 is running a cycle, the master or arbiter drops BHLDA (Figure 5-2B), the 82385 will complete the current cycle, move to BTH and remove the BHOLD request. If the 82385 still had cycles to run when it was kicked off the bus, it will immediately assert a new BHOLD and move to BTI to await bus acknowledgement. Note, however, that it will only move to BTI if BHLDA is negated, ensuring that the handshake sequence is completed.

There are several cases in which a slave 82385 will not immediately release the bus if BHLDA is dropped. For example, if BHLDA is dropped during a BT2P state, the 82385 has already committed to the next system bus pipelined cycle and will execute it before releasing the bus. Also, the 82385 will complete the second half of a two-cycle 16-bit transfer, or will complete a sequence of locked cycles before releasing the bus. This should not present any problems, as a properly designed arbiter will not assume that the 82385 has released the bus until it sees BHOLD become inactive.

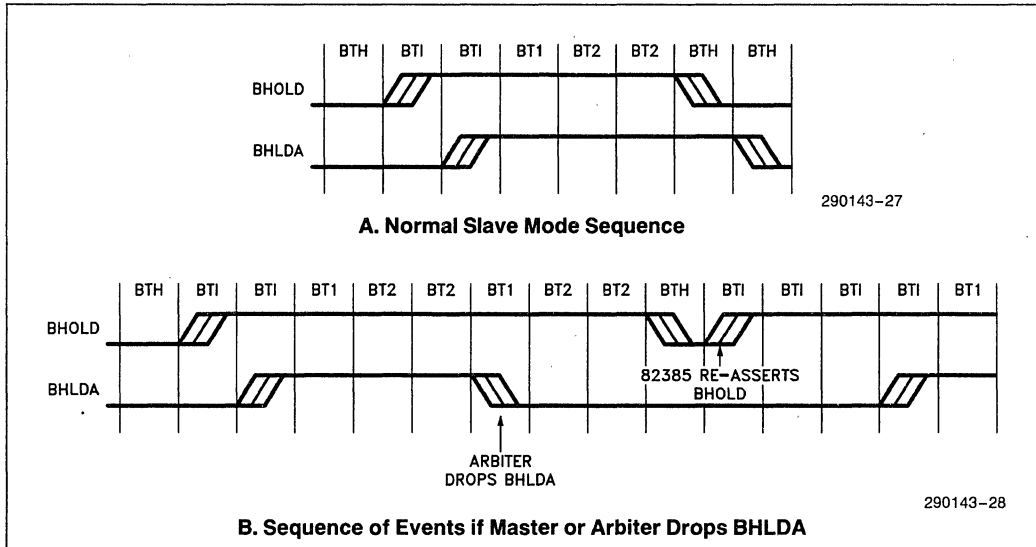


Figure 5-2. BHOLD/BHLDA—Slave Mode

5.2 The 82385 Local Bus

The 82385 bus can be broken up into two groups of signals: those which have direct 386 DX counterparts, and additional status and control signals provided by the 82385. The operation and interaction of all 82385 bus signals are depicted in Figures 5-3A through 5-3L for a wide variety of cycle sequences. These diagrams serve as a reference for the 82385 bus discussion and provide insight into the dual bus operation of the 82385.

5.2.1 82385 Bus Counterparts to 386 DX Signals

The following sections discuss the signals presented on the 82385 local bus which are functional equivalents to the signals present at the 386 DX local bus.

5.2.1.1 ADDRESS BUS (BA2-BA31) AND CYCLE DEFINITION SIGNALS (BM/IO#, BD/C#, BW/R#)

These signals are not driven directly by the 82385, but rather are the outputs of the 74374 address/cycle definition latch. (Refer to Figure 4-1 for the hardware interface.) This latch is controlled by the 82385 BACP and BAOE# outputs. The behavior and timing of these outputs and the latch they control (typically F or AS series TTL) ensure that BA2-BA31, BM/IO#, BW/R#, and BD/C# are compatible in timing and function to their 386 DX counterparts.

The behavior of BACP can be seen in Figure 5-3B, where the rising edge of BACP latches and forwards the 386 DX address and cycle definition signals in a BT1 or first BT2P state. However, the 82385 need not be the current bus master to latch the 386 DX address, as evidenced by cycle 4 of Figure 5-3A. In this case, the address is latched in frame 8, but not forwarded to the system (via BAOE#) until frame 10. (The latch and output enable functions of the 74374 are independent and invisible to one another.)

Note that in frames 2 and 6 the BACP pulses are marked "False." The reason is that BACP is issued and the address latched before the hit/miss determination is made. This ensures that should the cycle be a miss, the 82385 bus can move directly into BT1 without delay. In the case of a hit, the latched address is simply never qualified by the assertion of BADS#. The 82385 bus stays in BT1 if there is no access pending (new cycle is a hit) and no bus activity. It will move to and stay in BT2 if the system has requested a pipelined cycle and the 82385 does not have a pending bus access (new cycle is a hit).

5.2.1.2 DATA BUS (BD0-BD31)

The 82385 data bus is the system side of the 74646 latching transceiver. (See Figure 4-1.) This device is controlled by the 82385 outputs LDSTB, DOE#, and BT/R#. LDSTB latches data in write cycles, DOE# enables the transceiver outputs, and BT/R# controls the transceiver direction. The interaction of these signals and the transceiver is such that BD0-BD31 behave just like their 386 DX counterparts. The transceiver is configured such that data flow in write cycles (A to B) is latched, and data flow in read cycles (B to A) is flow-through.

Although BD0-BD31 function just like their 386 DX counterparts, there is a timing difference that must be accommodated for in a system design. As mentioned above, the transceiver is transparent during read cycles, so the transceiver propagation delay must be added to the 386 DX data setup. In addition, the cache SRAM setup must be accommodated for in cache read miss cycles.

For non-cacheable reads the data setup is given by:

$$\begin{array}{rcl} \text{Min BD0-BD31} & - & \text{386 DX Min} & + & \text{74646 B-to-A} \\ \text{Read Data Setup} & & \text{Data Setup} & & \text{Max Propagation} \\ & & & & \text{Delay} \end{array}$$

The required BD0–BD31 setup in a cache read miss is given by:

$$\begin{aligned} \text{Min BD0–BD31} &= 74646 \text{ B-to-A} & + & \text{Cache SRAM} \\ \text{Read Data} & \text{ Max Propagation} & & \text{Min Write} \\ \text{Setup} & \text{ Delay} & & \text{Setup} \\ & + \text{One CLK2} & - & \text{82385 CWEA\# or} \\ & \text{Period} & & \text{CWEB\# Min Delay} \end{aligned}$$

If a data buffer is located between the 386 DX data bus and the cache SRAMs, then its maximum propagation delay must be added to the above formula as well. A design analysis should be completed for every new design to determine actual margins.

A design can accommodate the increased data setup by choosing appropriately fast main memory DRAMs and data buffers. Alternatively, a designer may deal with the longer setup by inserting an extra wait state into cache read miss cycles. If an additional state is to be inserted, the system bus controller should sample the 82385 MISS# output to distinguish read misses from cycles that do not require the longer setup. Tips on using the 82385 MISS# signal are presented later in this chapter.

The behavior of LDSTB, DOE#, and BT/R# can be understood via Figures 5-3A through 5-3L. Note that in cycle 1 of Figure 5-3A (a non-cacheable system read), DOE# is activated midway through BT1, but in cycle 1 of Figure 5-3B (a cache read miss), DOE# is not activated until midway through BT2. The reason is that in a cacheable read cycle, the cache SRAMs are enabled to drive the 386 DX data bus before the outcome of the hit/miss decision (in anticipation of a hit). In cycle 1 of Figure 5-3B, the assertion of DOE# must be delayed until after the 82385 has disabled the cache output buffers. The result is that N=0 main memory should not be mapped into the cache.

5.2.1.3 BYTE ENABLES (BBE0#–BBE3#)

These outputs are driven directly by the 82385, and are completely compatible in timing and function with their 386 DX counterparts. When a 386 DX cycle is forwarded to the 82385 bus, the 386 DX byte enables are duplicated on BBE0#–BBE3#. The one exception is a cache read miss, during which BBE0#–BBE3# are all active regardless of the status of the 386 DX byte enables. This ensures that the cache is updated with a valid 32-bit entry.

5.2.1.4 ADDRESS STATUS (BADS#)

BADS# is identical in function and timing to its 386 DX counterpart. It is asserted in BT1 and BT2P states, and indicates that valid address and cycle definition (BA2–BA31, BBE0#–BBE3#, BM/IO#, BW/R#, BD/C#) information is available on the 82385 bus.

5.2.1.5 READY (BREADY#)

The 82385 BREADY# input terminates 82385 bus cycles just as the 386 DX READY# input terminates 386 DX bus cycles. The behavior of BREADY# is the same as that of READY#, but note in the A.C. timing specifications that a cache read miss requires a longer BREADY# setup than do other cycles. This must be accommodated for in ready logic design.

5.2.1.6 NEXT ADDRESS (BNA#)

BNA# is identical in function and timing to its 386 DX counterpart. Note that in Figures 5-3A through 5-3L, BNA# is assumed asserted in every BT1P or first BT2 state. Along with the 82385's pipelining of the 386 DX, this ensures that the timing diagrams accurately reflect the full pipelined nature of the dual bus structure.

5.2.1.7 BUS LOCK (BLOCK#)

The 386 DX flags a locked sequence of cycles by asserting LOCK#. During a locked sequence, the 386 DX does not acknowledge hold requests, so the sequence executes without interruption by another master. The 82385 forces all locked 386 DX cycles to run on the 82385 bus (unless LBA# is active), regardless of whether or not the referenced location resides in the cache. In addition, a locked sequence of 386 DX cycles is run as a locked sequence on the 82385 bus; BLOCK# is asserted and the 82385 does not allow the sequence to be interrupted. Locked writes (hit or miss) and locked read misses affect the cache and cache directory just as their unlocked counterparts do. A locked read hit, however, is handled differently. The read is necessarily

forced to run on the 82385 local bus, and as the data returns from main memory, it is "re-copied" into the cache. (See Figure 5-3L.) The directory is not changed as it already indicates that this location exists in the cache. This activity is invisible to the system and ensures that semaphores are properly handled.

BLOCK# is asserted during locked 82385 bus cycles just as LOCK# is asserted during locked 386 DX cycles. The BLOCK# maximum valid delay, however, differs from that of LOCK#, and this must be accounted for in any circuitry that makes use of BLOCK#. The difference is due to the fact that LOCK#, unlike the other 386 DX cycle definition signals, is not pipelined. The situation is clarified in Figure 5-3K. In cycle 2 the state of LOCK# is not known before the corresponding system read starts (Frames 4 and 5). In this case, LOCK# is asserted at the beginning of T1P, and the delay for BLOCK# to become active is the delay of LOCK# from the 386 DX plus the propagation delay through the 82385. This occurs because T1P and the corresponding BT1P are concurrent (Frame 5). The result is that BLOCK# should not be sampled at the end of BT1P. The first appropriate sampling point is midway through the next state, as shown in Frame 6. In Figure 5-3L, the maximum delay for BLOCK# to become valid in Frame 4 is the same as the maximum delay for LOCK# to become valid from the 386 DX. This is true since the pipelining issue discussed above does not occur.

The 82385 should negate BLOCK# after BREADY# of the last 82385 Locked Cycle was asserted and Lock turns inactive. This means that in a sequence of cycles which begins with a 82385 Locked Cycle and goes on with all the possible Locked Cycles (other 82385 cycles, idles, and local cycles), while LOCK# is continuously active, the 82385 will maintain BLOCK# active continuously. Another implication is that in a Locked Posted Write Cycle followed by non-locked sequence, BLOCK# is negated one CLK after BREADY# of the write cycle. In other 82385 Locked Cycles, followed by non-locked sequences, BLOCK# is negated one CLK after LOCK# is negated, which occurs two CLKs after BREADY# is asserted. In the last case BLOCK# active moves by one CLK to the non-locked sequence.

The arbitration rules of Locked Cycles are:

MASTER MODE:

BHOLD input signal is ignored when BLOCK# or internal lock (16-bit non-aligned cycle) are active. BHLDA output signal remains inactive, and BAOE# output signal remains active at that time interval.

SLAVE MODE:

The 82385 does not relinquish the system bus if BLOCK# or internal lock are active. The BHOLD output signal remains active when BLOCK# or internal lock is active plus one CLK. The BHLDA input signal is ignored when BLOCK# or the internal lock is active plus one CLK. This means the 82385 slave does not respond to BHLDA inactivation. The BAOE# output signal remains active during the same time interval.

5.2.2 Additional 82385 Bus Signals

The 82385 bus provides two status outputs and one control input that are unique to cache operation and thus have no 386 DX counterparts. The outputs are MISS#, and WBS, and the input is FLUSH.

5.2.2.1 CACHE READ/WRITE MISS INDICATION (MISS#)

MISS# can be thought of as an extra 82385 bus cycle definition signal similar to BM/IO#, BW/R#, and BD/C#, that distinguishes cacheable read and write misses from other cycles. MISS#, like the other definition signals, becomes valid with BADS# (BT1 or first BT2P). The behavior of MISS# is illustrated in Figures 5-3B, 5-3C, and 5-3J. The 82385 floats MISS# when another master owns the bus, allowing multiple 82385s to share the same node in multi-cache systems. MISS# should thus be lightly pulled up ($\sim 20\text{ K}\Omega$) to keep it negated during hold (BTH) states.

MISS# can serve several purposes. As discussed previously, the BD0-BD31 and BREADY# setup times in a cache read miss are longer than in other cycles. A bus controller can distinguish these cycles by gating MISS# with BW/R#. MISS# may also prove useful in gathering 82385 system performance data.

5.2.2.2 WRITE BUFFER STATUS (WBS)

WBS is activated when 386 DX write cycle data is latched into the 74646 latching transceiver (via LDSTB). It is deactivated upon completion of the write cycle on the 82385 bus when the 82385 sees the BREADY# signal. WBS behavior is illustrated in Figures 5-3F through 5-3J, and potential applications are discussed in Chapter 3.

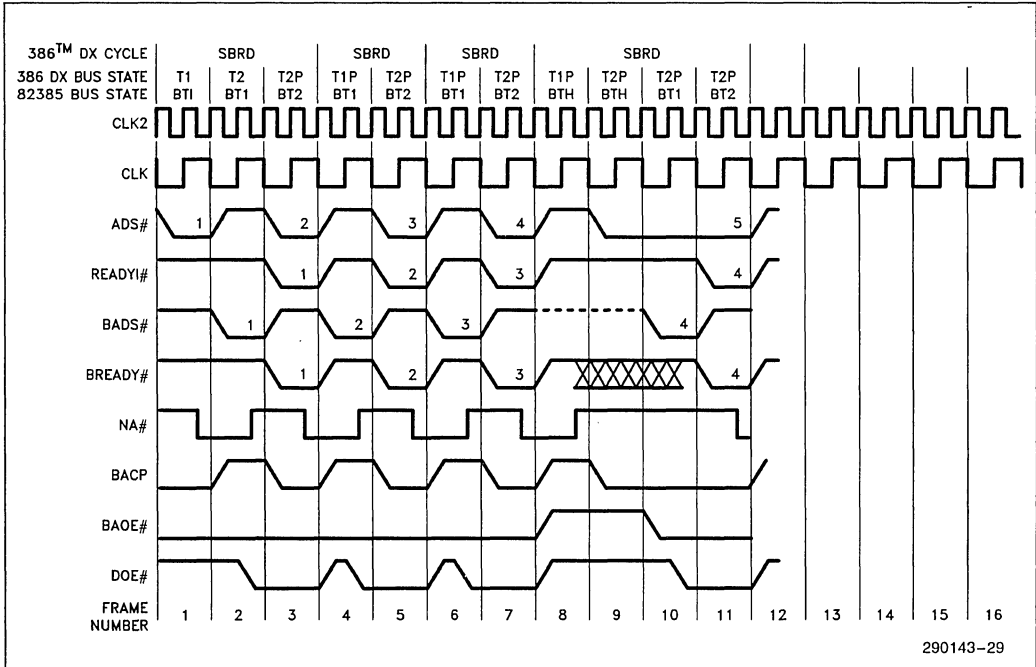


Figure 5-3A. Consecutive SBRD Cycles—(N = 0)

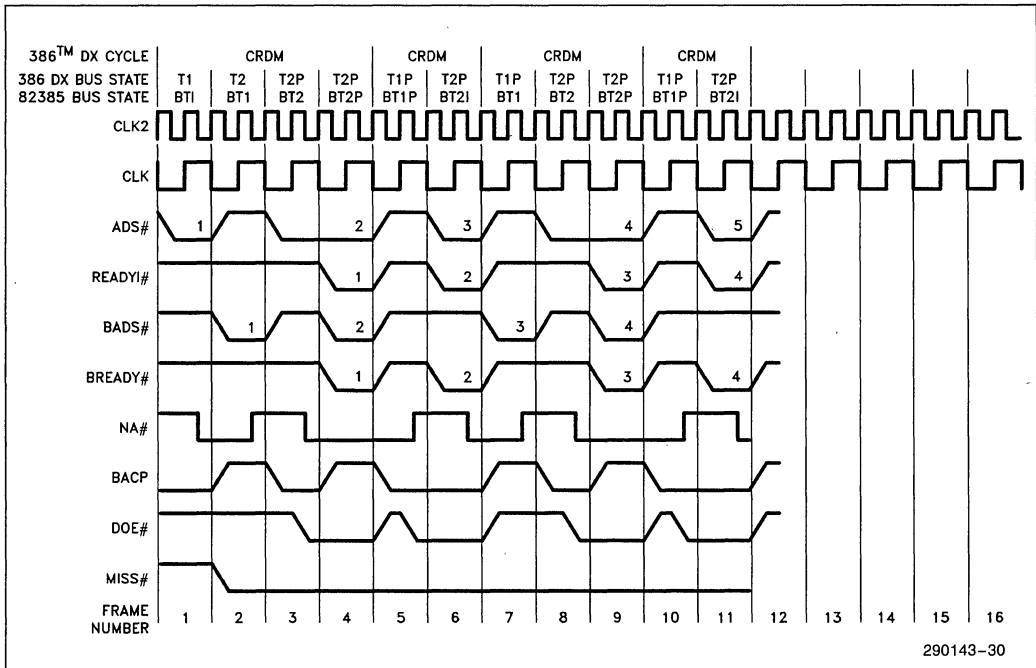


Figure 5-3B. Consecutive CRDM Cycles—(N = 1)

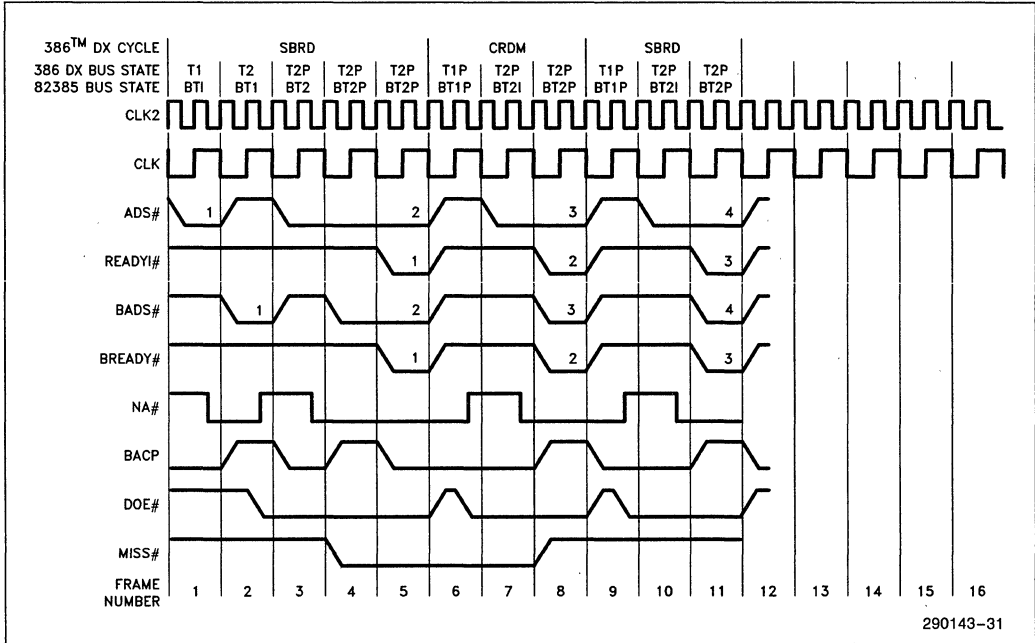


Figure 5-3C. SBRD, CRDM, SBRD—(N = 2)

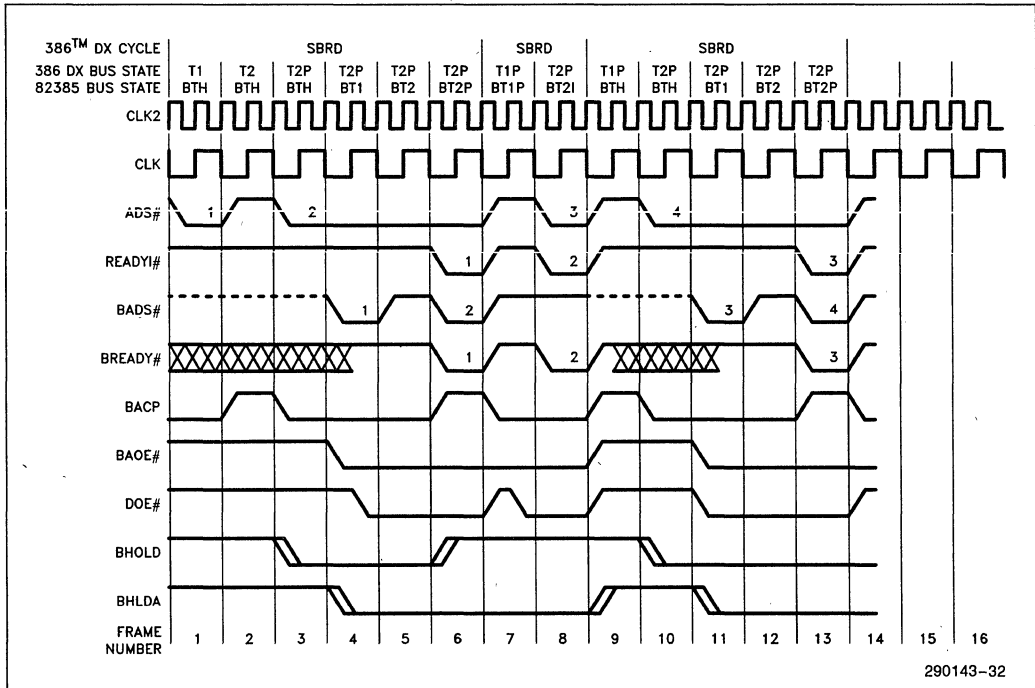
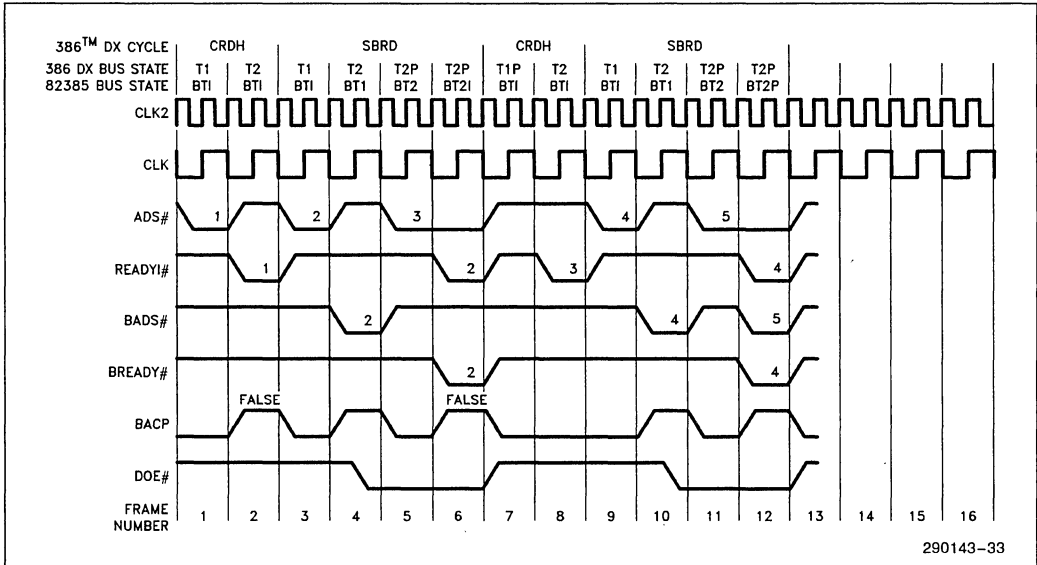
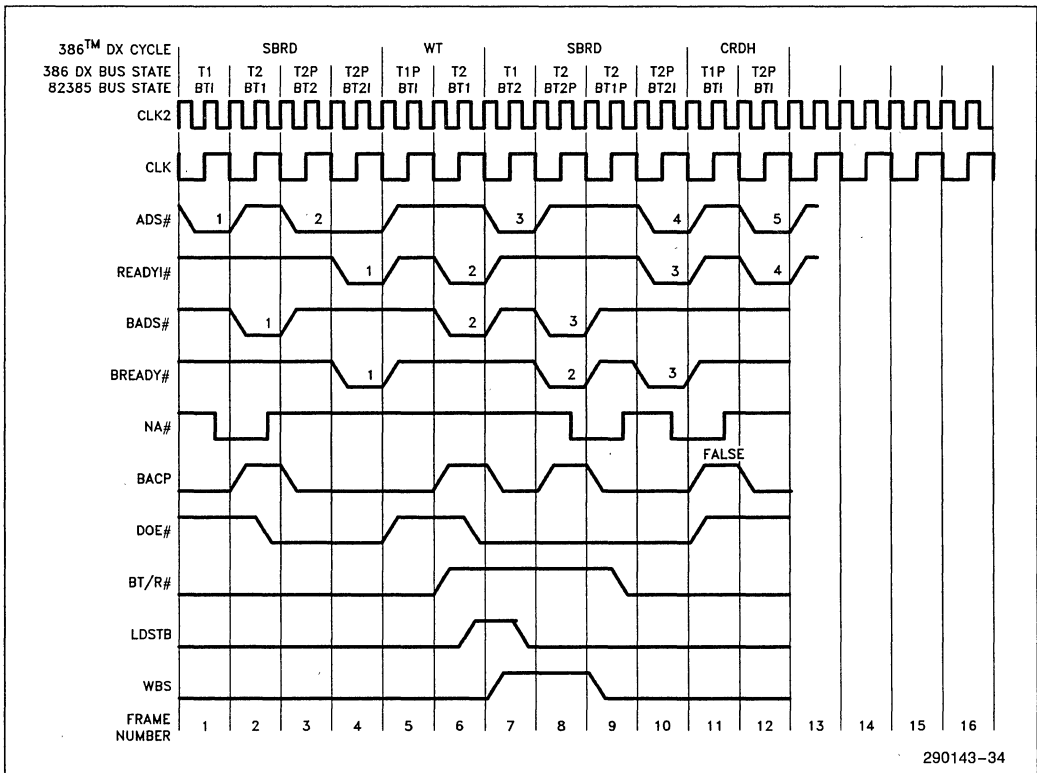


Figure 5-3D. SBRD Cycles Interleaved with BTH States—(N = 1)



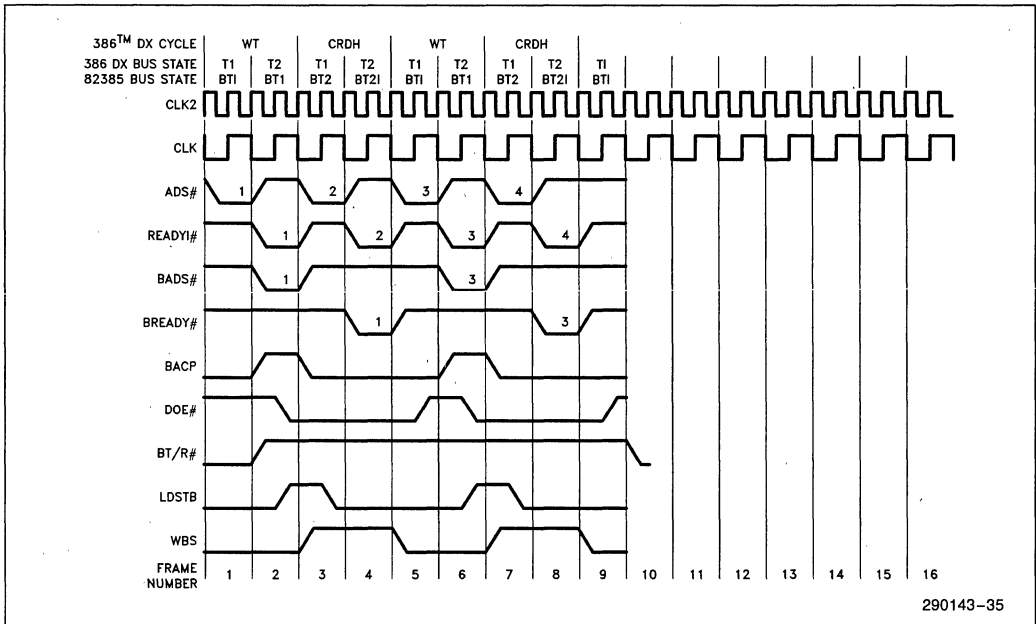
290143-33

Figure 5-3E. Interleaved SBRD/CRDH Cycles—(N = 1)



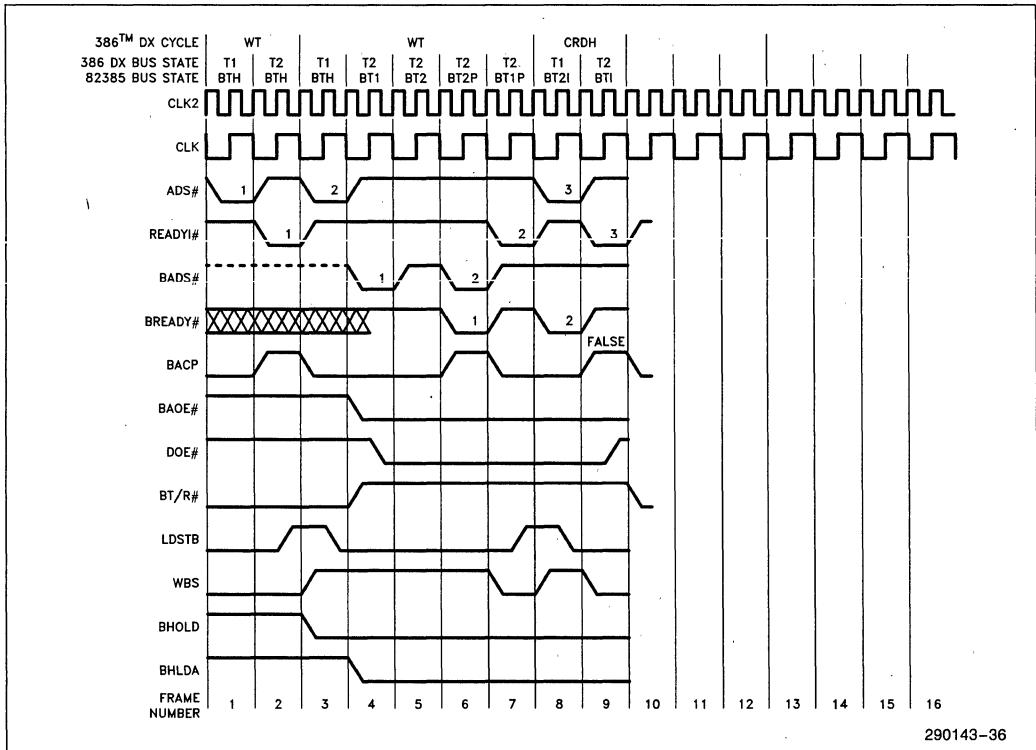
290143-34

Figure 5-3F. SBRD, WT, SBRD, CRDH—(N = 1)



290143-35

Figure 5-3G. Interleaved WT/CRDH Cycles—(N = 1)



290143-36

Figure 5-3H. WT, WT, CRDH—(N = 1)

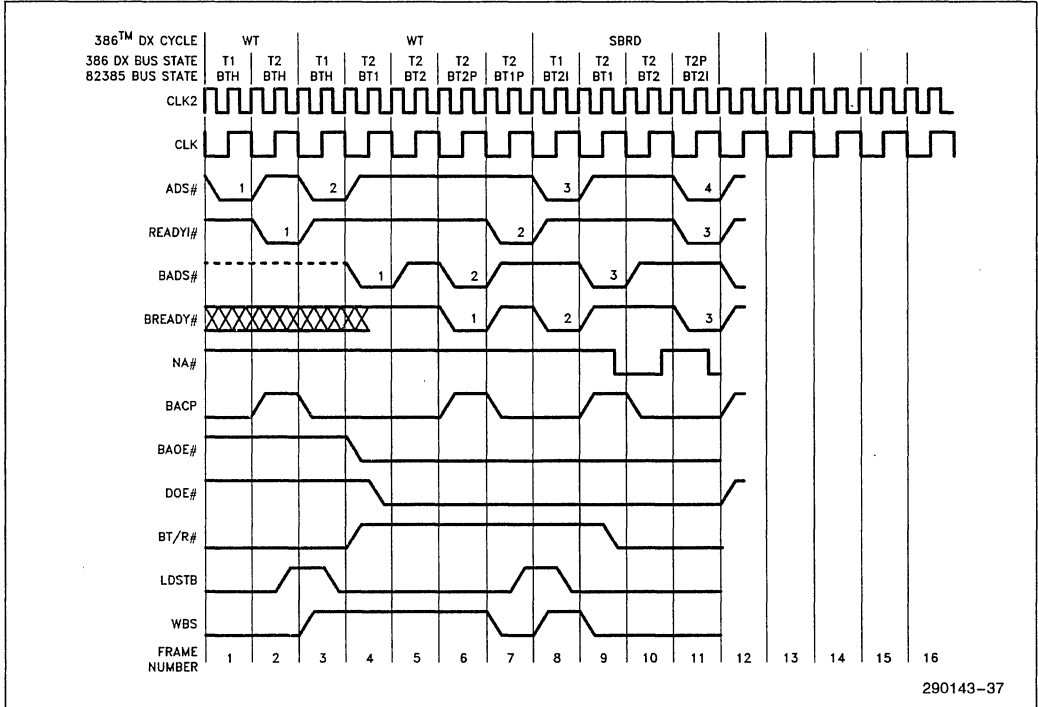


Figure 5-3I. WT, WT, SBRD—(N = 1)

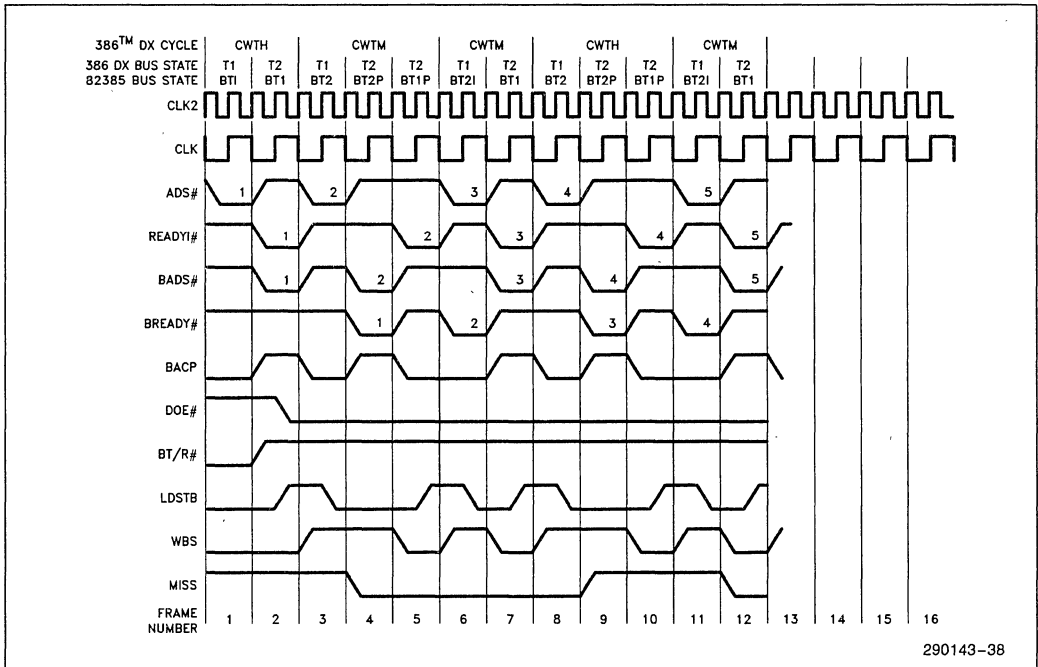


Figure 5-3J. Consecutive Write Cycles—(N = 1)

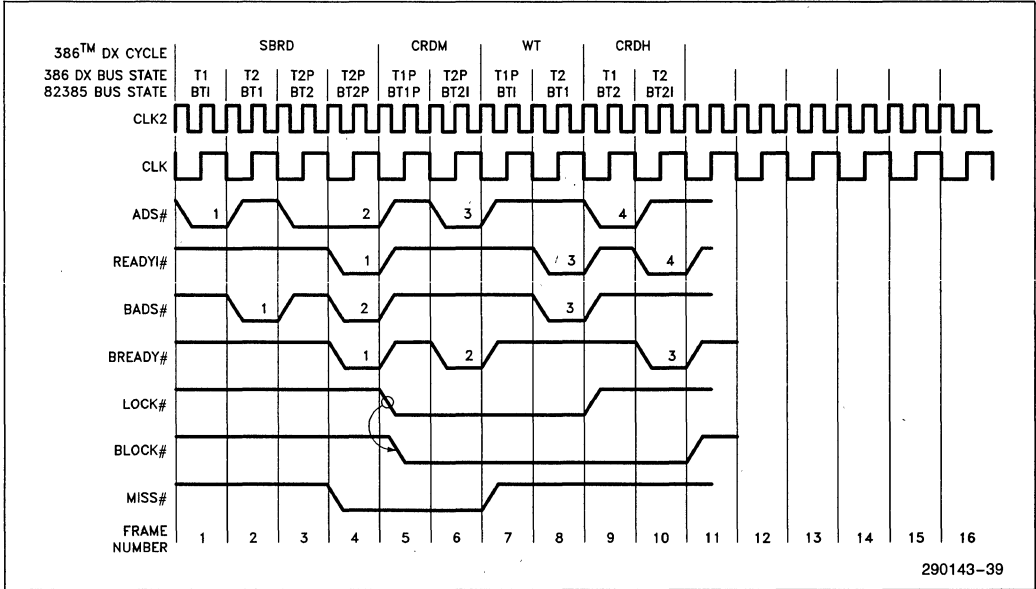


Figure 5-3K. LOCK # /BLOCK # in Non-Cacheable or Miss Cycles—(N = 1)

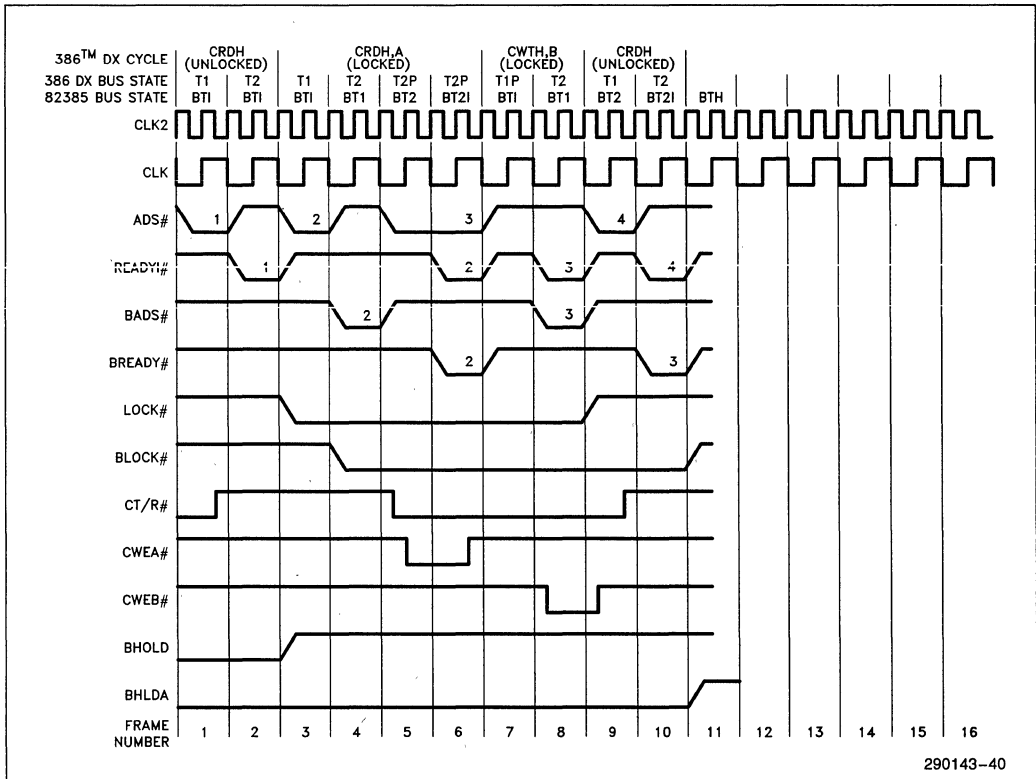


Figure 5-3L. LOCK # /BLOCK # in Cache Read Hit Cycle—(N = 1)

5.2.2.3 CACHE FLUSH (FLUSH)

FLUSH is an 82385 input which is used to reset all tag valid bits within the cache directory. The FLUSH input must be kept active for at least 4 CLK (8 CLK2) periods to complete the directory flush. Flush is generally used in diagnostics but can also be used in applications where snooping cannot guarantee coherency.

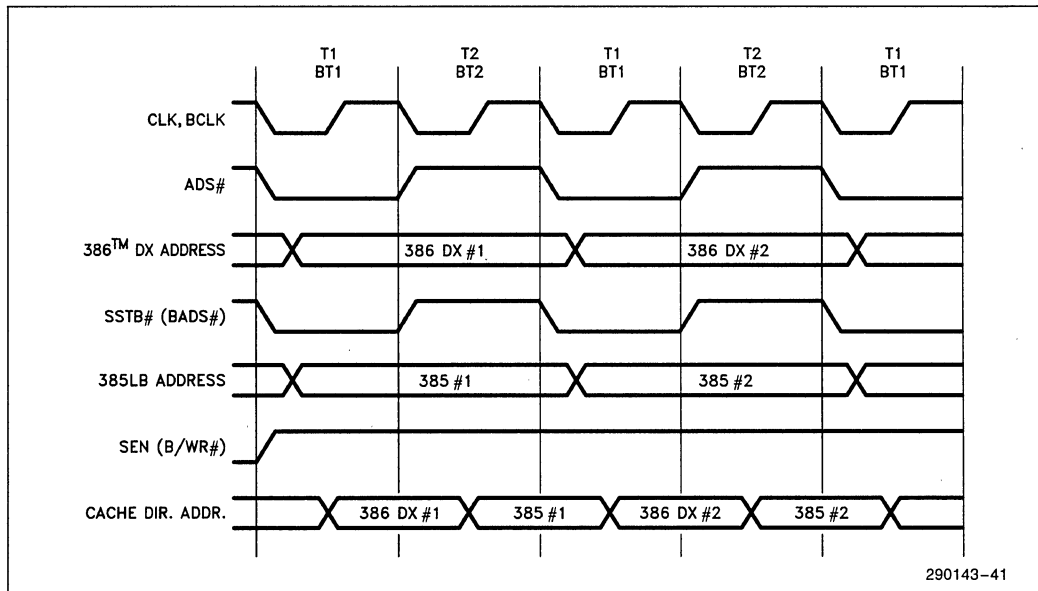
5.3 BUS WATCHING (SNOOP) INTERFACE

The 82385's bus watching interface consists of the snoop address (SA2-SA31), snoop strobe (SSTB#), and snoop enable (SEN) inputs. If masters reside at the system bus level, then the SA2-SA31 inputs are connected to the system address lines and SEN the system bus memory write command. SSTB# indicates that a valid address is present on the system bus. Note that the snoop bus inputs are synchronous, so care must be taken to ensure that they are stable during their sample windows. If no master resides beyond the 82385 bus level, then SA2-SA31, SEN, and SSTB# can respectively tie directly to BA2-BA31, BW/R#, and BADS#. However, it is recommended that SEN be driven by the logical "AND" of BW/R# and BM/IO# so as to prevent I/O writes from unnecessarily invalidating cache data.

When the 82385 detects a system write by another master, it internally latches SA2-SA31 and runs a cache look-up to see if the altered main memory location is duplicated in the cache. If yes (a snoop hit), the line valid bit associated with that cache entry is cleared. An important feature of the 82385 is that even if the 386 DX is running zero wait state hits out of the cache, all snoops are serviced. This is accomplished by time multiplexing the cache directory between the 386 DX address and latched system address. If the SSTB# signal occurs during an 82385 comparison cycle (for the 386 DX), the 386 DX cycle has the highest priority in accessing the cache directory. This takes the first of the two 386 DX states. The other state is then used for the snoop comparison. This worst case example, depicted in Figure 5-4, shows the 386 DX running zero wait state hits on the 386 DX local bus, and another master running zero wait state writes on the 82385 bus. No snoops are missed, and no performance penalty incurred.

5.4 RESET DEFINITION

Table 5-1 summarizes the states of all 82385 outputs during reset and initialization. A slave mode 82385 tri-states its "386 DX-like" front end. A master mode 82385 emits a pulse stream on its BACP output. As the 386 DX address and cycle definition lines reach their reset values, this stream will latch the reset values through to the 82385 bus.



290143-41

Figure 5.4. Interleaved Snoop and 386 DX Accesses to the Cache Directory

Table 5-1. Pin State During RESET and Initialization

Output Name	Signal Level During RESET and Initialization	
	Master Mode	Slave Mode
NA#	High	High
READY0#	High	High
BRDYEN#	High	High
CALEN	High	High
CWEA# - CWEB#	High	High
CS0# - CS3#	Low	Low
CT/R#	High	High
COEA# - COEB#	High	High
BADS#	High	High Z
BBE0# - BBE3#	386 DX BE#	High Z
BLOCK#	High	High Z
MISS#	High	High Z
BACP	Pulse ⁽¹⁾	Pulse
BAOE#	Low	High
BT/R#	Low	Low
DOE#	High	High
LDSTB	Low	Low
BHOLD	—	Low
BHLDA	Low	—
WBS	Low	Low

NOTE:

1. In Master Mode, BAOE# is low and BACP emits a pulse stream during reset. As the 386 DX address and cycle definition signals reach their reset values, the pulse stream on BACP will latch these values through to the 82385 local bus.

6.0 82385 SYSTEM DESIGN CONSIDERATIONS

6.1 INTRODUCTION

This chapter discusses techniques which should be implemented in an 82385 system. Because of the high frequencies and high performance nature of the 386 DX CPU/82385 system, good design and layout techniques are necessary. It is always recommended to perform a complete design analysis on new system designs.

6.2 POWER AND GROUNDING

6.2.1 Power Connections

The PGA 82385 utilizes 8 power (V_{CC}) and 10 ground (V_{SS}) pins. The PQFP 82385 has 9 power and 9 ground pins. All V_{CC} and V_{SS} pins must be connected to their appropriate plane. On a printed circuit board, all V_{CC} pins must be connected to the power plane and all V_{SS} pins must be connected to the ground plane.

6.2.2 Power Decoupling

Although the 82385 itself is generally a "passive" device in that it has few output signals, the cache

subsystem as a whole is quite active. Therefore, many decoupling capacitors should be placed around the 82385 cache subsystem.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the decoupling capacitors and their respective devices as much as possible. Capacitors specifically for PGA packages are also commercially available, for the lowest possible inductance.

6.2.3 Resistor Recommendations

Because of the dual bus structure of the 82385 subsystem (386 DX Local Bus and 82385 Local Bus), any signals which are recommended to be pulled up will be respective to one of the busses. The following sections will discuss signals for both busses.

6.2.3.1 386 DX LOCAL BUS

For typical designs, the pullup resistors shown in Table 6-1 are recommended. This table correlates to Chapter 7 of the 386 DX Data Sheet. However, particular designs may have a need to differ from the listed values. Design analysis is recommended to determine specific requirements.

6.2.3.2 82385 LOCAL BUS

Pullup resistor recommendations for the 82385 Local Bus signals are shown in Table 6-2. Design analysis is necessary to determine if deviations to the typical values given is needed.

Table 6-1. Recommended Resistor Pullups to V_{CC} (386 DX Local Bus)

Pin and Signal	Pullup Value	Purpose
ADS# PGA E13 PQFP 123	20 K Ω \pm 10%	Lightly Pull ADS# Negated for 386 DX Hold States
LOCK# PGA F13 PQFP 118	20 K Ω \pm 10%	Lightly Pull LOCK# Negated for 386 DX Hold States

Table 6-2. Recommended Resistor Pullups to V_{CC} (82385 Local Bus)

Signal and Pin	Pullup Value	Purpose
BADS# PGA N9 PQFP 89	20 K Ω \pm 10%	Lightly Pull BADS# Negated for 82385 Hold States
BLOCK# PGA P9 PQFP 86	20 K Ω \pm 10%	Lightly Pull BLOCK# Negated for 82385 Hold States
MISS# PGA N8 PQFP 85	20 K Ω \pm 10%	Lightly Pull MISS# Negated for 82385 Hold States

6.3 82385 SIGNAL CONNECTIONS

6.3.1 Configuration Inputs

The 82385 configuration signals (M/S#, 2W/D#, DEFOE#) must be connected (pulled up) to the appropriate logic level for the system design. There is also a reserved 82385 input which must be tied to the appropriate level. Refer to Table 6-3 for the signals and their required logic level.

Table 6-3. 82385 Configuration Inputs Logic Levels

Pin and Signal	Logic Level	Purpose
M/S# PGA B13 PQFP 129	High	Master Mode Operation
	Low	Slave Mode Operation
2W/D# PGA D12 PQFP 127	High	2-Way Set Associative
	Low	Direct Mapped
Resrvd PGA L14 PQFP 102	High	Must be tied to V_{CC} via a pull-up for proper functionality
DEFOE# PGA A14 PQFP 128	N/A	Define Cache Output Enables. Allows use of any SRAM.

NOTE:

The listed 82385 pins which need to be tied high should use a pull-up resistor in the range of 5 K Ω to 20 K Ω .

6.3.2 CLK2 and RESET

The 82385 has two inputs to which the 386 DX CLK2 signal must be connected. One is labeled CLK2 (82385 PGA pin C13, PQFP lead 126) and the other is labeled BCLK2 (82385 PGA pin L13, PQFP lead 103). These two inputs must be tied together on the printed circuit board.

The 82385 also has two reset inputs. RESET (82385 PGA pin D13, PQFP lead 125) and BRESET (82385 PGA pin K12, PQFP lead 104) must be connected on the printed circuit board.

6.4 UNUSED PIN REQUIREMENTS

For reliable operation, ALWAYS connect unused inputs to a valid logic level. As is the case with most other CMOS processes, a floating input will increase the current consumption of the component and give an indeterminate state to the component.

6.5 CACHE SRAM REQUIREMENTS

The 82385 offers the option of using SRAMs with or without an output enable pin. This is possible by inserting a transceiver between the SRAMs and the 386 DX local data bus and strapping DEFOE# to the appropriate logic level for a given system configuration. This transceiver may also be desirable in a system which has a very heavily loaded 386 DX local data bus. The following sections discuss the SRAM requirements for all cache configurations.

6.5.1 Cache Memory without Transceivers

As discussed in Section 3.2, the 82385 presents all of the control signals necessary to access the cache memory. The SRAM chip selects, write enables, and output enables are driven directly by the 82385. Table 6-4 lists the required SRAM specifications. These specifications allow for zero margins. They should be used as guides for the actual system design.

6.5.2 Cache Memory With Transceivers

To implement an 82385 subsystem using cache memory transceivers, COEA# or COEB# must be used as output enable signals for the transceivers and DEFOE# must be appropriately strapped for proper COEx# functionality (since the cache SRAM transceivers must be enabled for writes as well as reads). DEFOE# must be tied high when using cache SRAM transceivers. In a 2-way set associative organization, COEA# enables the transceiver for bank A and COEB# enables the bank B transceiver. A direct mapped cache may use either COEA# or COEB# to enable the transceiver. Table 6-5 lists the required SRAM specifications. These specifications allow for zero margin. They should be used as guides for the actual system design.

Table 6-4. SRAM Specs for Non-Buffered Cache Memory

	SRAM Spec Requirements					
	Direct Mapped			2-Way Set Associative		
	20	25	33	20	25	33
Read Cycle Requirements						
Address Access (MAX)	44	36	27	42	34	27
Chip Select Access (MAX)	56	44	35	56	41	35
OE# to Data Valid (MAX)	19	13	10	14	13	10
OE# to Data Float (MAX)	20	15	10	20	15	10
Write Cycle Requirements						
Chip Select to End of Write (MIN)	30	25	20	30	25	20
Address Valid to End of Write (MIN)	42	37	29	40	37	29
Write Pulse Width (MIN)	30	25	20	30	25	20
Data Setup (MAX)	—	—	—	—	—	—
Data Hold (MIN)	4	4	2	4	4	2

Table 6-5. SRAM Specs for Buffered Cache Memory

SRAM Spec Requirements						
	Direct Mapped			2-Way Set Associative		
	20	25	33	20	25	33
Read Cycle Requirements						
Address Access (MAX)	37	29	20	35	29	20
Chip Select Access (MAX)	48	36	27	48	36	27
OE# to Data Valid (MAX)	N/A	N/A	N/A	N/A	N/A	N/A
OE# to Data Float (MAX)	N/A	N/A	N/A	N/A	N/A	N/A
Write Cycle Requirements						
Chip Select to End of Write (MIN)	30	25	20	30	23	20
Address Valid to End of Write (MIN)	42	37	29	40	36	27
Write Pulse Width (MIN)	30	25	20	30	25	20
Data Setup (MAX)	15	10	10	15	10	10
Data Hold (MIN)	3	3	3	3	3	3

7.0 SYSTEM TEST CONSIDERATIONS

7.1 INTRODUCTION

Power On Self Testing (POST) is performed by most systems after a reset. This chapter discusses the requirements for properly testing an 82385 based system after power up.

7.2 MAIN MEMORY (DRAM) TESTING

Most systems perform a memory test by writing a data pattern and then reading and comparing the data. This test may also be used to determine the total available memory within the system. Without properly taking into account the 82385 cache memory, the memory test can give erroneous results. This will occur if the cache responds with read hits during the memory test routine.

7.2.1 Memory Testing Routine

In order to properly test main memory, the test routine must not read from the same block consecutively. For instance, if the test routine writes a data pattern to the first 32 kbytes of memory (0000-7FFFH), reads from the same block, writes a new pattern to the same locations (0000-7FFFH), and reads the new pattern, the second pattern tested would have had data returned from the 82385 cache memory. Therefore, it is recommended that the test routine work with a memory block of at least 64 kbytes. This will guarantee that no 32 kbyte block will be read twice consecutively.

7.3 82385 CACHE MEMORY TESTING

With the addition of SRAMs for the cache memory, it may be desirable for the system to be able to test the cache SRAMs during system diagnostics. This requires the test routine to access only the cache memory. The requirements for this routine are based on where it resides within the memory map. This can be broken into two areas: the routine residing in cacheable memory space or the routine residing in either non-cacheable memory or on the 386 DX local bus (using the LBA# input).

7.3.1 Test Routine in the NCA# or LBA# Memory Map

In this configuration, the test routine will never be cached. The recommended method is code which will access a single 32 kbyte block during the test. Initially, a 32 kbyte read (assume 0000-7FFFH) must be executed. This will fill the cache directory with the address information which will be used in the diagnostic procedure. Then, a 32 kbyte write to the same address locations (0000-7FFFH) will load the cache with the desired test pattern (due to write hits). The comparison can be made by completing another 32 kbyte read (same locations, 0000-7FFFH), which will be cache read hits. Subsequent writes and reads to the same addresses will enable various patterns to be tested.

7.3.2 Test Routine in Cacheable Memory

In this case, it must be understood that the diagnostic routine must reside in the cache memory before the actual data testing can begin. Otherwise, when the 386 DX performs a code fetch, a location within the cache memory which is to be tested will be altered due to the read miss (code fetch) update.

The first task is to load the diagnostic routine into the top of the cache memory. It must be known how much memory is required for the code as the rest of the cache memory will be tested as in the earlier method. Once the diagnostics have been cached (via read updates), the code will perform the same type of read/write/read/compare as in the routine explained in the above section. The difference is that now the amount of cache memory to be tested is 32 kbytes minus the length of the test routine.

7.4 82385 CACHE DIRECTORY TESTING

Since the 82385 does not directly access the data bus, it is not possible to easily complete a comparison of the cache directory. (The 82385 can serially transmit its directory contents. See Section 7.5.) However, the cache memory tests described in Section 7.3 will indicate if the directory is working properly. Otherwise, the data comparison within the diagnostics will show locations which fail.

There is a slight possibility that the cache memory comparison could pass even if locations within the directory gave false hit/miss results. This could cause the comparison to always be performed to main memory instead of the cache and give a proper comparison to the 386 DX. The solution here is to use the MISS# output of the 82385 as an indicator to a diagnostic port which can be read by the 386 DX. It could also be used to flag an interrupt if a failure occurs.

The implementation of these techniques in the diagnostics will assure proper functionality of the 82385 subsystem.

7.5 SPECIAL FUNCTION PINS

As mentioned in Chapter 3, there are three 82385 pins which have reserved functions in addition to their normal operational functions. These pins are MISS#, WBS, and FLUSH.

As discussed previously, the 82385 performs a directory flush when the FLUSH input is held active for at least 4 CLK (8 CLK²) cycles. However, the FLUSH pin also serves as a diagnostic input to the 82385. The 82385 will enter a reserved mode if the FLUSH pin is high at the falling edge of RESET.

If, during normal operation, the FLUSH input is active for only one CLK (2 CLK²) cycle/s, the 82385 will enter another reserved mode. Therefore it must be guaranteed that FLUSH is active for at least the 4 CLK (8 CLK²) cycle specification.

WBS and MISS# serve as outputs in the 82385 reserved modes.

8.0 MECHANICAL DATA

8.1 INTRODUCTION

This chapter discusses the physical package and its connections in detail.

8.2 PIN ASSIGNMENT

The 82385 pinout as viewed from the top side of the component is shown by Figure 8-1. Its pinout as viewed from the Pin side of the component is shown in Figure 8-2.

V_{CC} and V_{SS} connections must be made to multiple V_{CC} and V_{SS} (GND) pins. Each V_{CC} and V_{SS} must be connected to the appropriate voltage level. The circuit board should include V_{CC} and GND planes for power distribution and all V_{CC} and V_{SS} pins must be connected to the appropriate plane.

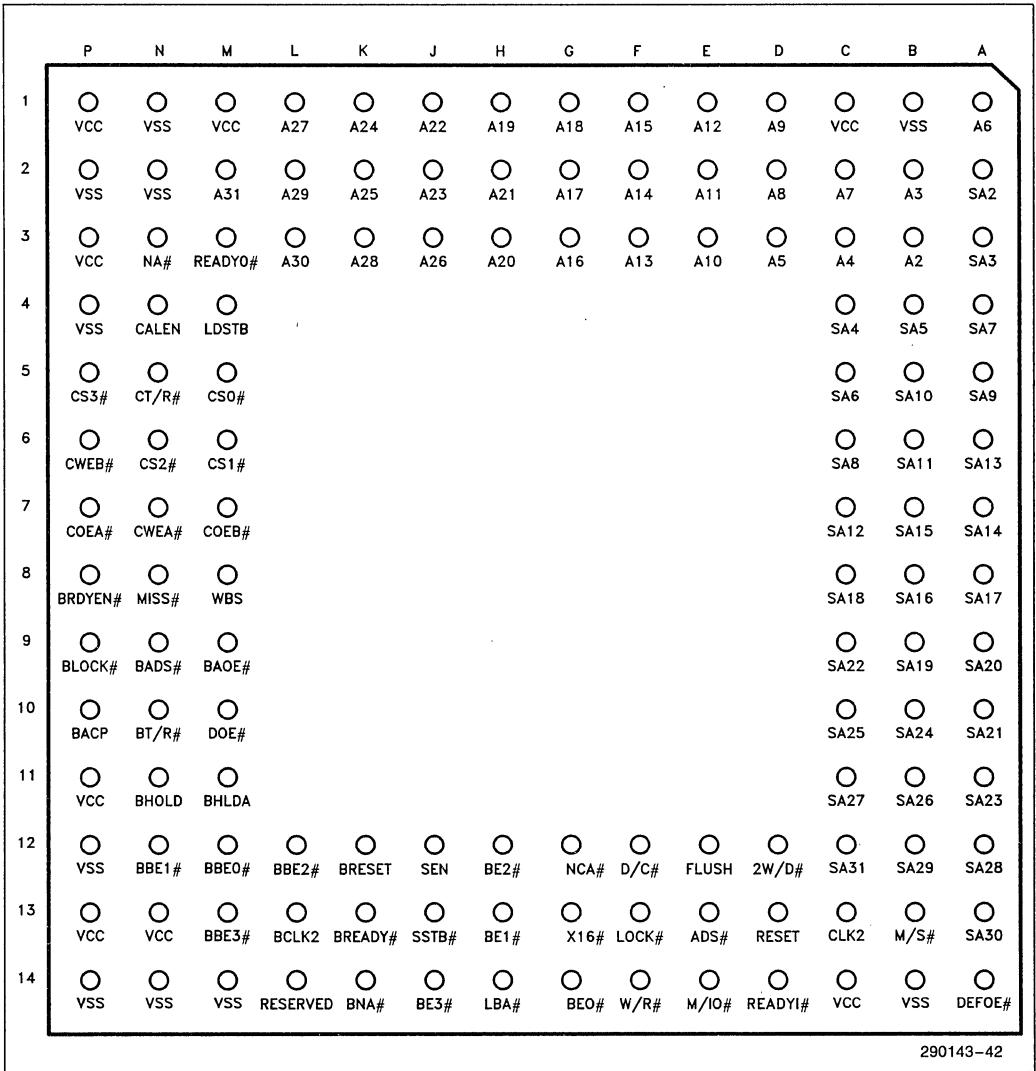


Figure 8-1. 82385 PGA Pinout—View from TOP Side

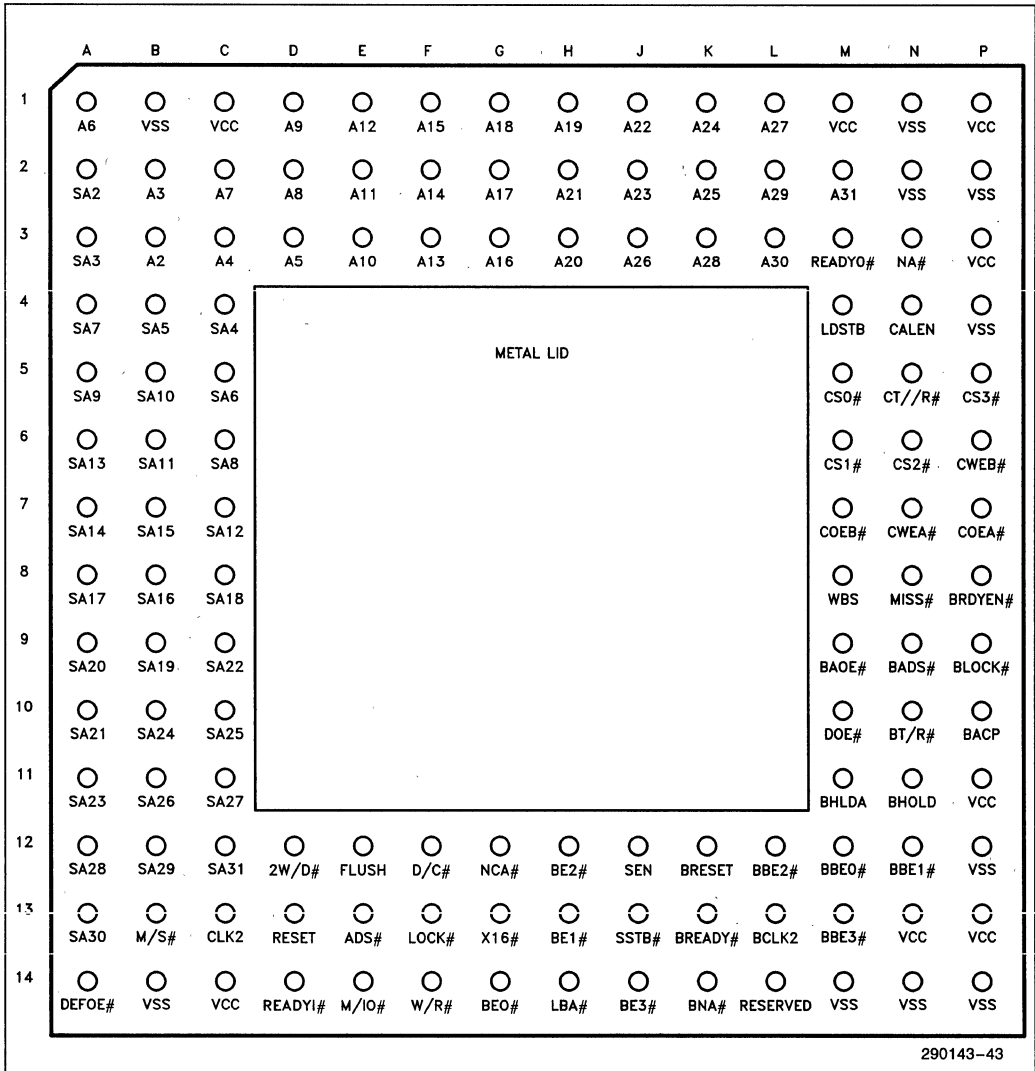


Figure 8-2. 82385 PGA Pinout—View from PIN Side

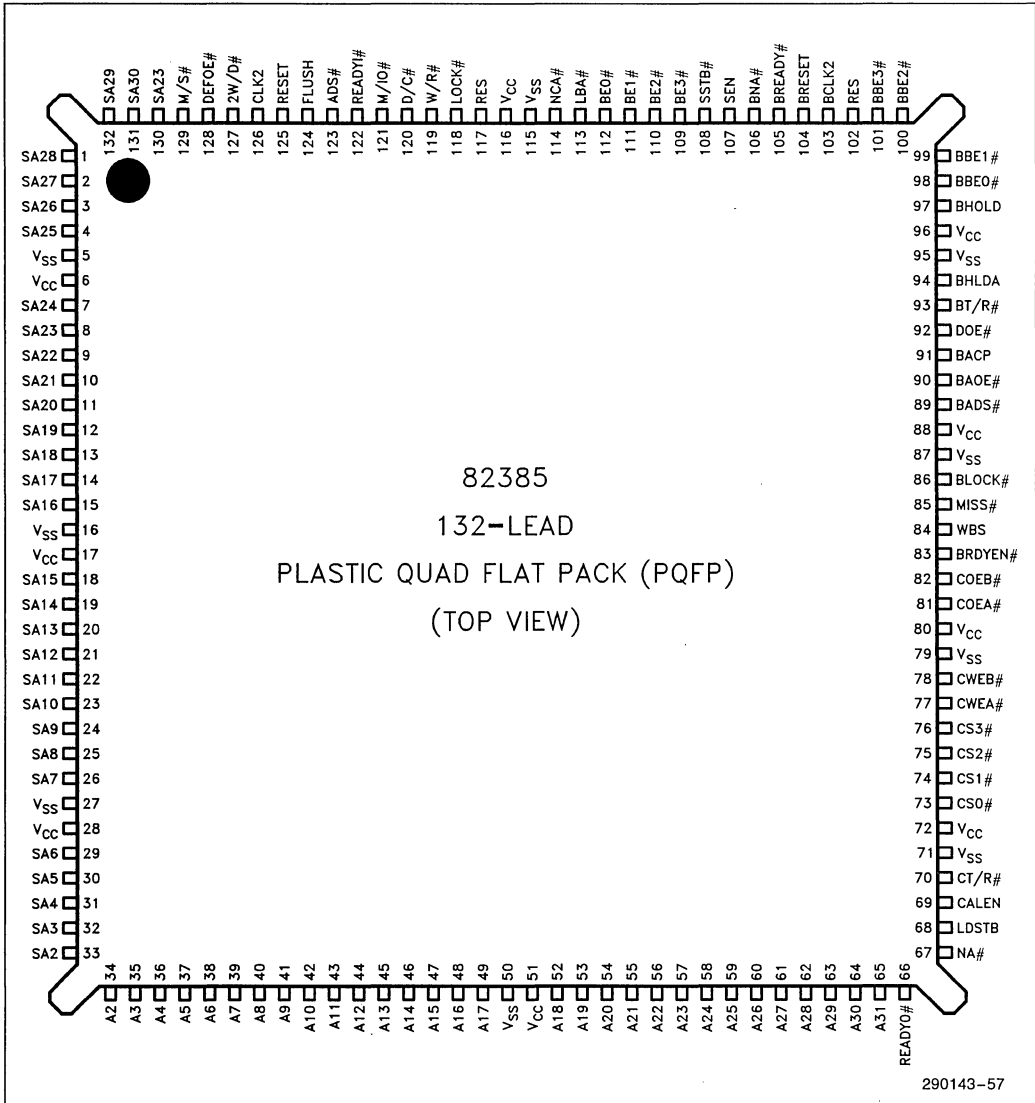


Figure 8-3. 82385 PGA Pinout—View from TOP Side

Table 8-1. 82385 PGA Pinout—Functional Grouping

PGA	PQFP	Signal
M2	65	A31
L3	64	A30
L2	63	A29
K3	62	A28
L1	61	A27
J3	60	A26
K2	59	A25
K1	58	A24
J2	57	A23
J1	56	A22
H3	54	A20
H1	53	A19
G1	52	A18
G2	49	A17
G3	48	A16
F1	47	A15
F2	46	A14
F3	45	A13
E1	44	A12
E2	43	A11
E3	42	A10
D1	41	A9
D2	40	A8
C2	39	A7
A1	38	A6
D3	37	A5
C3	36	A4
B2	35	A3
B3	34	A2
G14	112	BE0#
H13	111	BE1#
H12	110	BE2#
J14	109	BE3#
C13	126	CLK2
D13	125	RESET
K12	104	BRESET
L13	103	BCLK2

PGA	PQFP	Signal
C12	130	SA31
A13	131	SA30
B12	132	SA29
A12	1	SA28
C11	2	SA27
B11	3	SA26
C10	4	SA25
B10	7	SA24
A11	8	SA23
C9	9	SA22
A10	10	SA21
A9	11	SA20
B9	12	SA19
C8	13	SA18
A8	14	SA17
B8	15	SA16
B7	18	SA15
A7	19	SA14
A6	20	SA13
C7	21	SA12
B6	22	SA11
B5	23	SA10
A5	24	SA9
C6	25	SA8
A4	26	SA7
C5	29	SA6
B4	30	SA5
C4	31	SA4
A3	32	SA3
A2	33	SA2
J12	107	SEN
J13	108	SSTB#
L14	102	RESERVED

PGA	PQFP	Signal
—	116	V _{CC}
C1	6	V _{CC}
C14	17	V _{CC}
M1	28	V _{CC}
N13	51	V _{CC}
P1	72	V _{CC}
P3	80	V _{CC}
P11	88	V _{CC}
P13	96	V _{CC}
E13	123	ADS#
F14	119	W/R#
F12	120	D/C#
E14	121	M/IO#
F13	118	LOCK#
N3	67	NA#
G13	117	X16#
G12	114	NCA#
H14	113	LBA#
D14	122	READYI#
M3	66	READYO#
E12	124	FLUSH
M8	84	WBS
N8	85	MISS#
A14	128	DEFOE#
D12	127	2W/D#
B13	129	M/S#
M10	92	DOE#
M4	68	LDSTB
N11	97	BHOLD
M11	94	BHLDA

PGA	PQFP	Signal
B1	5	V _{SS}
B14	16	V _{SS}
M14	27	V _{SS}
N1	50	V _{SS}
N2	71	V _{SS}
N14	79	V _{SS}
P2	87	V _{SS}
P4	95	V _{SS}
P12	115	V _{SS}
P14	—	V _{SS}
N9	89	BADS#
M12	98	BBE0#
N12	99	BBE1#
L12	100	BBE2#
M13	101	BBE3#
P9	86	BLOCK#
K14	106	BNA#
N4	69	CALEN
P7	81	COEA#
M7	82	COEB#
N7	77	CWEA#
P6	78	CWEB#
M5	73	CS0#
M6	74	CS1#
N6	75	CS2#
P5	76	CS3#
N5	70	CT/R#
P8	83	BRDYEN#
K13	105	BREADY#
P10	91	BACP
M9	90	BAOE#
N10	93	BT/R#

8.3 PACKAGE DIMENSIONS AND MOUNTING

The 82385 package is a 132-pin ceramic Pin Grid Array (PGA). The pins are arranged 0.100 inch (2.5 mm) center-to-center, in a 14 x 14 matrix, three rows around (Figure 8-3).

A wide variety of available sockets allow low insertion force or zero insertion force mounting. These come in a choice of terminals such as soldertail, surface mount, or wire wrap.

8.4 PACKAGE THERMAL SPECIFICATION

The PGA case temperature should be measured at the center of the top surface opposite the pins, as in Figure 8-4. The case temperature may be measured in any environment to determine whether or not the 82385 is within the specified operating range.

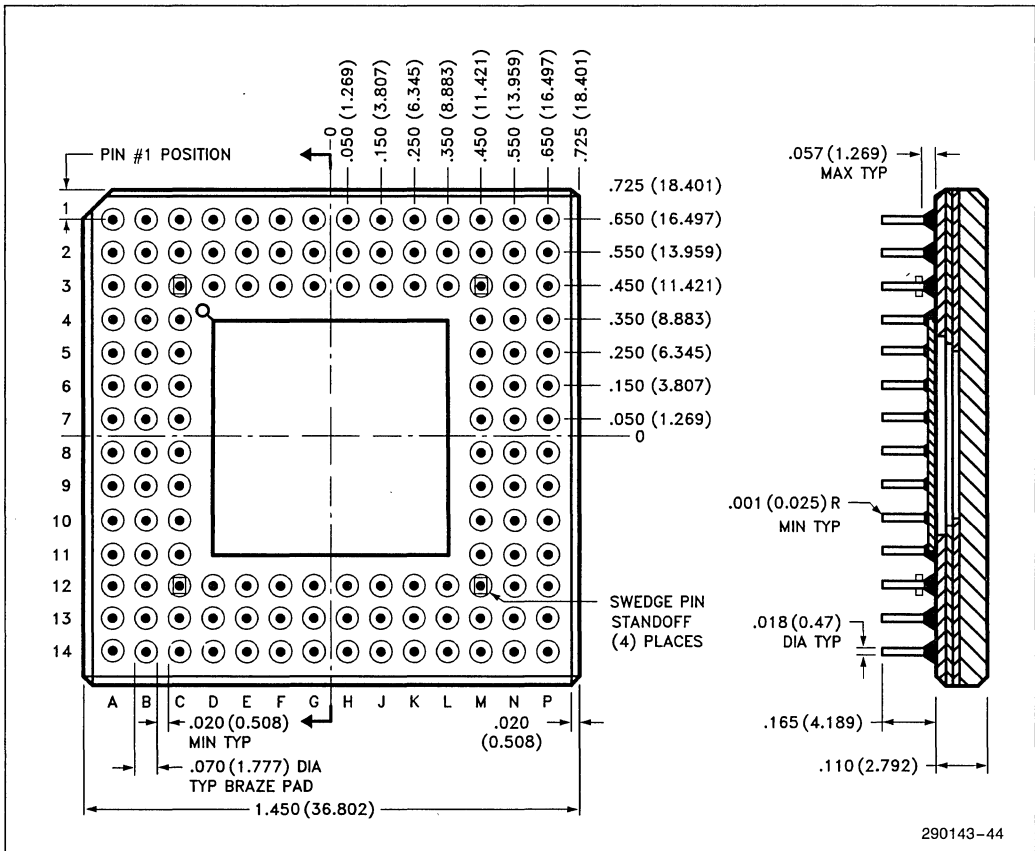


Figure 8-3.1. 132-Pin PGA Package Dimensions

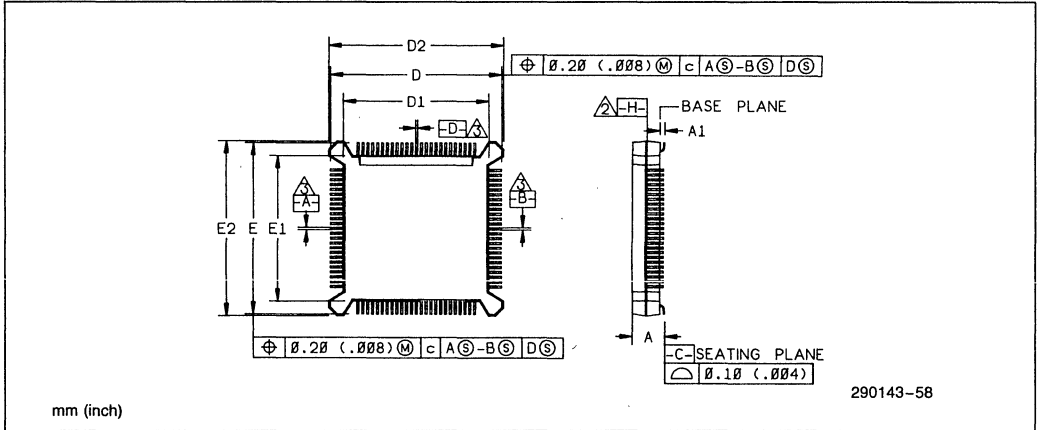


Figure 8-3.2. Principal Dimensions and Datums

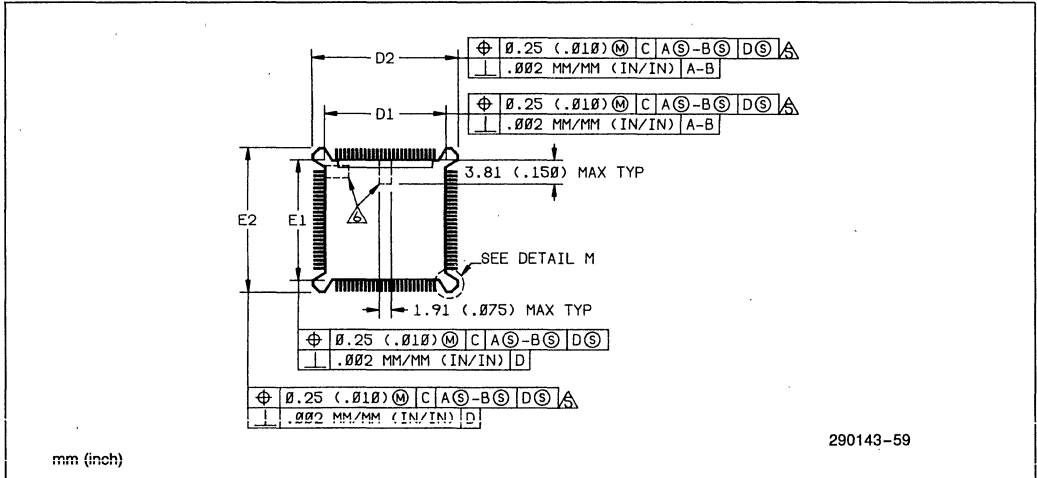


Figure 8-3.3. Molded Details

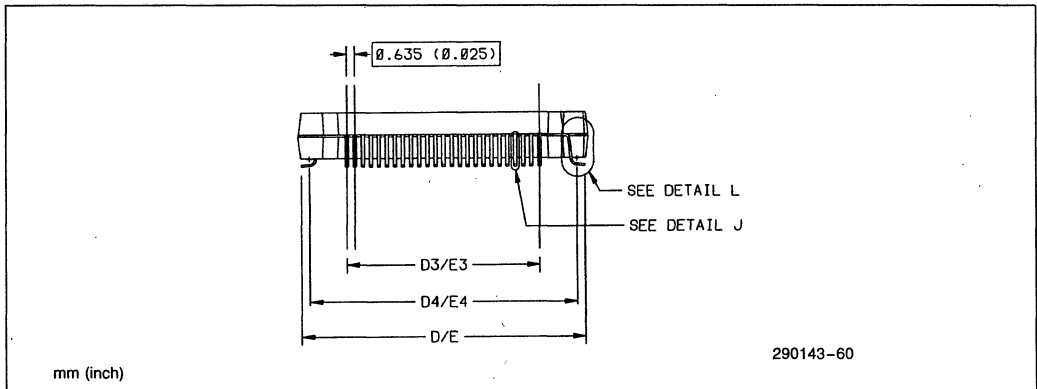


Figure 8-3.4. Terminal Details

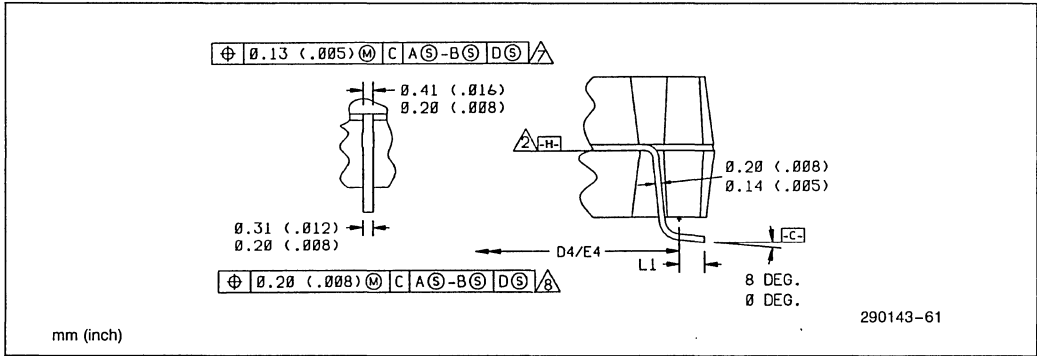


Figure 8-3.5. Typical Lead

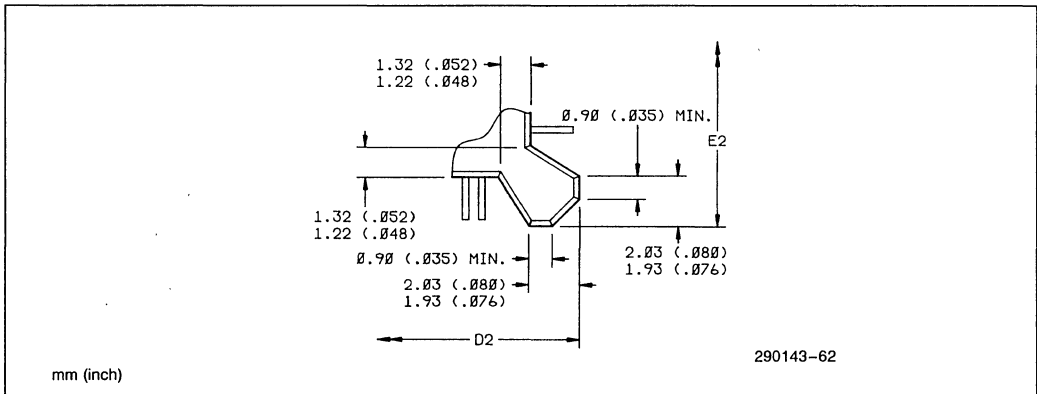


Figure 8-3.6. Detail M

PLASTIC QUAD FLAT PACK

Table 8-2. Symbol List for Plastic Quad Flat Pack

Letter or Symbol	Description of Dimensions
A	Package height: distance from seating plane to highest point of body
A1	Standoff: Distance from seating plane to base plane
D/E	Overall package dimension: lead tip to lead tip
D1/E1	Plastic body dimension
D2/E2	Bumper Distance
D3/E3	Footprint
L1	Foot length
N	Total number of leads

NOTES:

1. All dimensions and tolerances conform to ANSI Y14.5M-1982.
2. Datum plane -H- located at the mold parting line and coincident with the bottom of the lead where lead exits plastic body.
3. Datums A-B and -D- to be determined where center leads exit plastic body at datum plane -H-.
4. Controlling Dimension, Inch.
5. Dimensions D1, D2, E1 and E2 are measured at the mold parting line and do not include mode protrusion. Allowable mold protrusion of 0.18mm (0.007 in.) per side.
6. Pin 1 identifier is located within one of the two zones indicated.
7. Measured at datum plane -H-.
8. Measured at seating plane datum -C-.

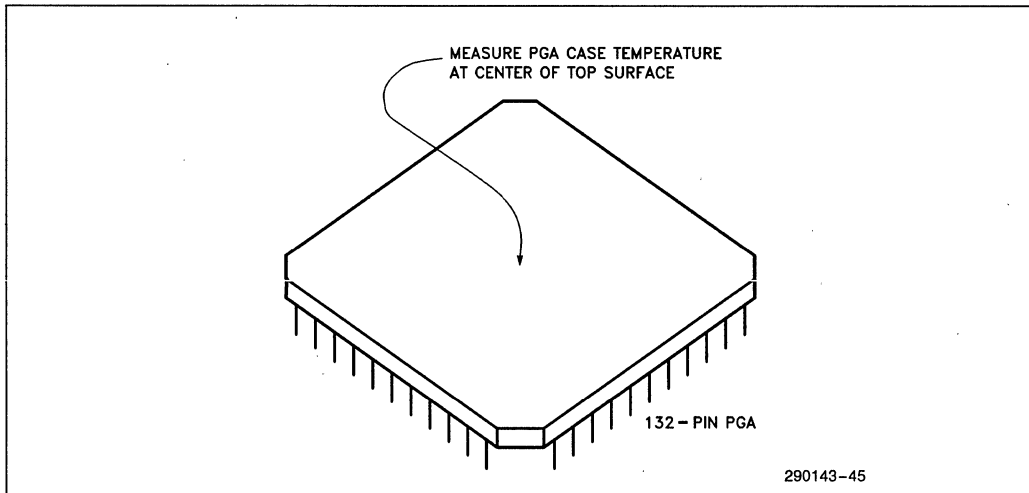


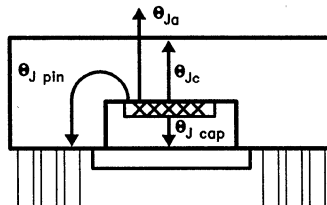
Figure 8-4. Measuring 82385 PGA Case Temperature

Table 8-3. 82385 PGA Package Typical Thermal Characteristics.

Parameter	Thermal Resistance—°C/Watt						
	Airflow—f ³ /min (m ³ /sec)						
	0 (0)	50 (0.25)	100 (0.50)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
θ Junction-to-Case (Case Measured as Figure 8.4)	2	2	2	2	2	2	2
θ Case-to-Ambient (No Heatsink)	19	18	17	15	12	10	9
θ Case-to-Ambient (with Omnidirectional Heatsink)	16	15	14	12	9	7	6
θ Case-to-Ambient (with Unidirectional Heatsink)	15	14	13	11	8	6	5

NOTES:

1. Table 8-3 applies to 82385 PGA plugged into socket or soldered directly onto board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{J-CAP} = 4^{\circ}\text{C/W}$ (approx.)
 $\theta_{J-PIN} = 4^{\circ}\text{C/W}$ (inner pins) (approx.)
 $\theta_{J-PIN} = 8^{\circ}\text{C/W}$ (outer pins) (approx.)



290143-46

Thermal Resistance—°C/Watt							
Parameter	Airflow—f ³ /min (m ³ /sec)						
	0 (0)	50 (0.25)	100 (0.50)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
θ Junction-to-Case (Case Measured as Figure 8.4)							
θ Case-to-Ambient (No Heatsink)							
θ Case-to-Ambient (with Omnidirectional Heatsink)							
θ Case-to-Ambient (with Unidirectional Heatsink)							

NOTES:

- Table 8-4 applies to 82385 PQFP plugged into socket or soldered directly onto board.
- $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
- $\theta_{J-CAP} = 4^{\circ}\text{C/W}$ (approx.)
 $\theta_{J-PIN} = 4^{\circ}\text{C/W}$ (inner pins) (approx.)
 $\theta_{J-PIN} = 8^{\circ}\text{C/W}$ (outer pins) (approx.)

9.0 ELECTRICAL DATA

9.1 INTRODUCTION

This chapter presents the A.C. and D.C. specifications for the 82385.

9.2 MAXIMUM RATINGS

Storage Temperature -65°C to +150°C

Case Temperature under Bias ... -65°C to +110°C

Supply Voltage with Respect

to V_{SS} -0.5V to +6.5V

Voltage on Any Other Pin -0.5V to $V_{CC} + 0.5V$

NOTE:

Stress above those listed may cause permanent damage to the device. This is a stress rating only and functional operation at these or any other conditions above those listed in the operational sections of this specification is not implied.

Exposure to absolute maximum rating conditions for extended periods may affect device reliability. Although the 82385 contains protective circuitry to resist damage from static electrical discharges, always take precautions against high static voltages or electric fields.

9.3 D.C. SPECIFICATIONS $V_{CC} = 5V \pm 5\%$; $V_{SS} = 0V$

Table 9-1. D.C. Specifications

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{IL}	Input Low Voltage	-0.3	0.8	V	(Note 1)
V_{IH}	Input High Voltage	2.0	$V_{CC} + 0.3$	V	
V_{CL}	CLK2, BCLK2 Input Low	-0.3	0.8	V	(Note 1)
V_{CH}	CLK2, BCLK2 Input High	3.7	$V_{CC} + 0.3$	V	
V_{OL}	Output Low Voltage		0.45	V	$I_{OL} = 4 \text{ mA}$
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -1 \text{ mA}$
I_{CC}	Supply Current		300	mA	(Note 2) (Note 4)
I_{LI}	Input Leakage Current		± 15	μA	$0V < V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		± 15	μA	$0.45 < V_{OUT} < V_{CC}$
C_{IN}	Input Capacitance		10	pF	(Note 3)
C_{OUT}	Output Capacitance		10	pF	(Note 3)
C_{CLK}	CLK2 Input Capacitance		15	pF	(Note 3)

NOTES:

1. Minimum value is not 100% tested.
2. I_{CC} is specified with inputs driven to CMOS levels. I_{CC} may be higher if driven to TTL levels.
3. Not 100% tested. Test conditions $f_C = 1 \text{ MHz}$, Inputs = 0V, $T_{CASE} = \text{Room}$.
4. 300 mA is the maximum I_{CC} at 33 MHz.
275 mA is the maximum I_{CC} at 25 MHz.
250 mA is the maximum I_{CC} at 20 MHz.

9.4 A.C. SPECIFICATIONS

The A.C. specifications given in the following tables consist of output delays and input setup requirements. The A.C. diagram's purpose is to illustrate the clock edges from which the timing parameters are measured. The reader should not infer any other timing relationships from them. For specific information on timing relationships between signals, refer to the appropriate functional section.

A.C. spec measurement is defined in Figure 9-1. Inputs must be driven to the levels shown when A.C. specifications are measured. 82385 output delays

are specified with minimum and maximum limits, which are measured as shown. 82385 input setup and hold times are specified as minimums and define the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 82385 operation.

9.4.1 Frequency Dependent Signals

The 82385 has signals whose output valid delays are dependent on the clock frequency. These signals are marked in the A.C. Specification Tables with a Note 1.

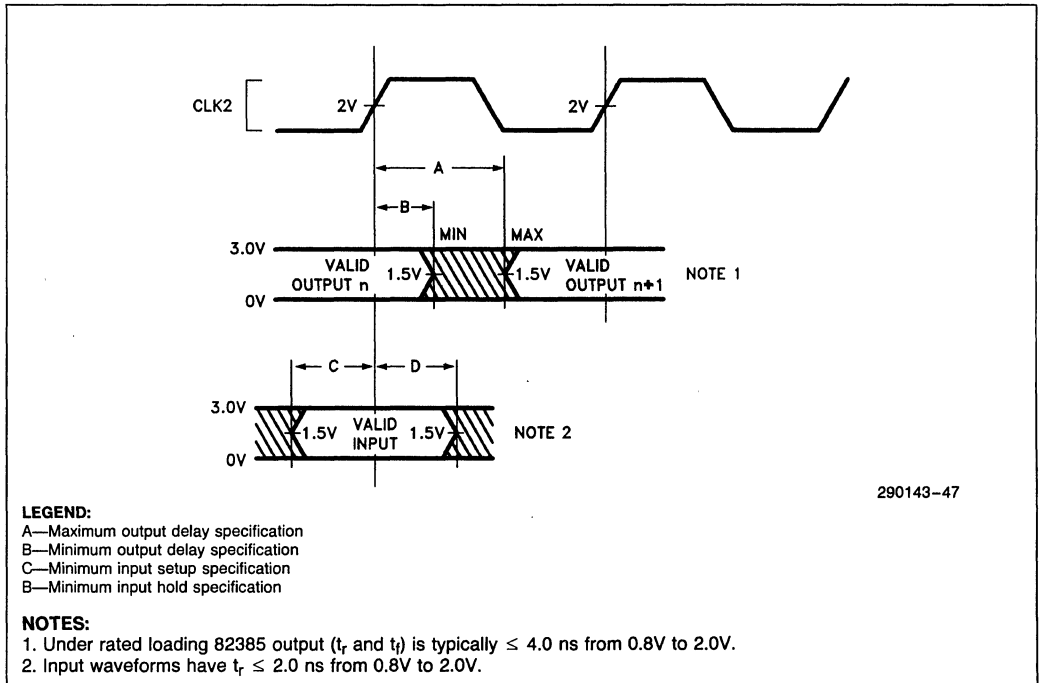


Figure 9-1. Drive Levels and Measurement Points for A.C. Specification

A.C. SPECIFICATION TABLES

Many of the A.C. Timing parameters are frequency dependent. The frequency dependent A.C. Timing parameters are guaranteed only at the maximum specified operating frequency.

Table 9-2. 82385 A.C. Timing Specifications

V_{CC} = 5.0 ±5%

Symbol	Parameter	20 MHz		25 MHz		33 MHz		Units	Notes
		Min	Max	Min	Max	Min	Max		
T _{CASE}	Case Temperature	0	85	0	75	0	75	°C	
t ₁	Operating Frequency	15.40	20.00	15.40	25.00	15.40	33.33	MHz	
t ₂	CLK2, BCLK2 Clock Period	25.00	32.50	20.00	32.50	15.00	32.50	ns	
t _{3a}	CLK2, BCLK2 High Time @ 2.0V	10		8		6.25		ns	
t _{3b}	CLK2, BCLK2 High Time @ 3.7V	7		5		4.5		ns	(Note 8)
t _{4a}	CLK2, BCLK2 Low Time @ 2.0V	10		8		6.25		ns	
t _{4b}	CLK2, BCLK2 Low Time @ 0.8V	8		6		4.5		ns	(Note 8)
t ₅	CLK2, BCLK2 Fall Time		8		7		4	ns	(Notes 8, 9)
t ₆	CLK2, BCLK2 Rise Time		8		7		4	ns	(Notes 8, 9)
t _{7a}	A2–A19, A21–A31 Setup Time	19		18		13		ns	(Note 1)
t _{7b}	LOCK # Setup Time	16		14		9.5		ns	(Note 1)
t _{7c}	BE(0–3) # Setup Time	19		14		10		ns	(Note 1)
t _{7d}	A20 Setup Time	13		13		9		ns	(Note 1)
t ₈	A2–A31, BE(0–3) # LOCK # Hold Time	3		3		3		ns	
t _{9a}	M/IO #, D/C # Setup Time	22		17		13		ns	(Note 1)
t _{9b}	W/R # Setup Time	22		18		13		ns	(Note 1)
t _{9c}	ADS # Setup Time	22		18		13.5		ns	(Note 1)
t ₁₀	ADS #, D/C #, M/IO #, W/R # Hold Time	5		3		3		ns	
t ₁₁	READYI # Setup Time	12		8		7		ns	(Note 1)
t ₁₂	READYI # Hold Time	4		4		3		ns	
t _{13a1}	NCA # Setup Time (See t55b2)	21		18		13		ns	(Note 6)
t _{13a2}	NCA # Setup Time (See t55b3)	16		13		9		ns	(Note 6)
t _{13b}	LBA # Setup Time	10		8		5.75		ns	
t _{13c}	X16 # Setup Time	10		7		5.5		ns	
t _{14a}	NCA # Hold Time	4		3		3		ns	
t _{14b}	LBA #, X16 # Hold Time	4		3		3		ns	
t ₁₅	RESET, BRESET Setup Time	12		10		8		ns	
t ₁₆	RESET, BRESET Hold Time	4		3		2		ns	
t ₁₇	NA # Valid Delay	15	34	4	27	4	19.2	ns	(25 pF Load) (Note 1)
t ₁₈	READYO # Valid Delay	4	28	4	22	3	15	ns	(25 pF Load) (Note 1)
t ₁₉	BRDYEN # Valid Delay	4	28	4	21	3	13	ns	

Table 9-2. 82385 A.C. Timing Specifications (Continued)

 $V_{CC} = 5.0 \pm 5\%$

Symbol	Parameter	20 MHz		25 MHz		33 MHz		Units	Notes
		Min	Max	Min	Max	Min	Max		
t21a1	CALEN Rising, PHI1	3	24	4	21	3	15	ns	
t21a2	CALEN Falling, PHI1	3	24	4	21	3	15	ns	
t21a3	CALEN Falling in T1P, PHI2	3	24	4	21	3	15	ns	
t21b	CALEN Rising Following CWTH Cycle	3	34	4	27	3	20	ns	(Note 1)
t21c	CALEN Pulse Width	10		10		10		ns	
t21d	CALEN Rising to CS# Falling	13		13		13		ns	
t22a1	CWEx# Falling, PHI1 (CWTH)	4	25	4	23	3	18	ns	(Note 1)
t22a2	CWEx# Falling, PHI2 (CRDM)	4	25	4	23	3	18	ns	(Note 1)
t22b	CWEx# Pulse Width	30		25		20		ns	(Notes 1, 2)
t22c1	CWEx# Rising, PHI1 (CWTH)	4	25	4	21	3	16	ns	(Note 1)
t22c2	CWEx# Rising, PHI2 (CRDM)	12	25	8	21	6	16	ns	(Note 1)
t23a	CS(0-3) # Rising	12	37	9	29	3	25	ns	(Note 1)
t23b	COEx# Falling to CS(0-3) # Falling	0		0		0		ns	(Note 1)
t24	CT/R# Valid Delay	12	38	9	30	3	22	ns	(Note 1)
t25a	COEx# Falling (Direct)	1	22	4	19.5	3	15	ns	(25 pF Load)
t25b	COEx# Falling (2-Way)	1	24.5	4	19.5	3	15	ns	(25 pF Load) (Note 1)
t25c1	COEx# Rising Delay @ $T_{CASE} = \text{Min}$	5	17	4	17.5	3	12	ns	(25 pF Load)
t25c2	COEx# Rising Delay @ $T_{CASE} = \text{Max}$	5	19	4	19.5	3	12	ns	(25 pF Load)
t25d	CWEx# Falling to COEx# Falling or CWEx# Rising to COEx# Rising when DEFOE# = V_{CC}	0	5	0	5	0	5	ns	(25 pF Load)
t26	CS(0-3) # Falling to CWEx# Rising	30		25		20		ns	(Notes 1, 2)
t27	CWEx# Falling to CS(0-3) # Falling	0		0		0		ns	
t28a	CWEx# Rising to CALEN Rising	0		0		2		ns	
t28b	CWEx# Rising to CS(0-3) # Falling	0		0		2		ns	
t31	SA(2-31) Setup Time	19		10		8		ns	
t32	SA(2-31) Hold Time	3		3		3		ns	
t33	BADS# Valid Delay	6	28	4	21	3	16	ns	(Note 1)
t34	BADS# Float Delay	6	30	4	30	4	25	ns	(Note 3)
t35	BNA# Setup Time	9		7		7		ns	
t36	BNA# Hold Time	15		4		2		ns	
t37	BREADY# Setup Time	26		20		13		ns	(Note 1)
t38	BREADY# Hold Time	4		3		2		ns	
t40a	BACP Rising Delay	4	20	4	16	2	12	ns	
t40b	BACP Falling Delay	4	22	4	20	2	18	ns	

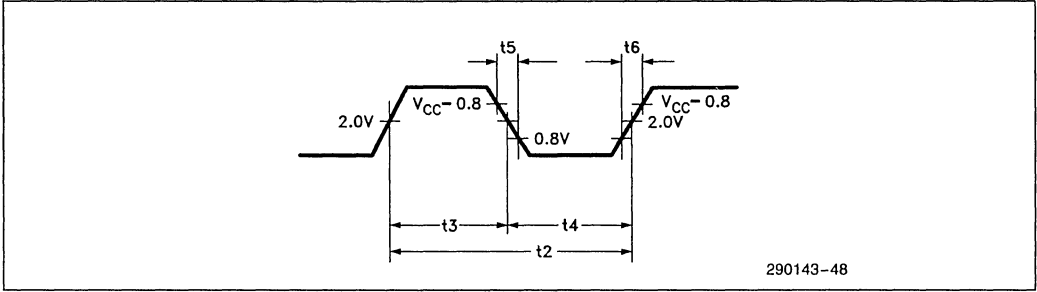
Table 9-2. 82385 A.C. Timing Specifications (Continued)

V_{CC} = 5.0 ± 5%

Symbol	Parameter	20 MHz		25 MHz		33 MHz		Units	Notes
		Min	Max	Min	Max	Min	Max		
t41	BAOE # Valid Delay	4	18	4	15	2	12	ns	
t43a	BT/R # Valid Delay	2	19	4	16	2	14	ns	
t43b1	DOE # Falling Delay	2	23	4	20	2	16	ns	
t43b2	DOE # Rising Delay @ T _{CASE} = Min	4	17	4	17	2	12	ns	
t43b3	DOE # Rising Delay @ T _{CASE} = Max	4	19	4	19	2	14	ns	
t43c	LDSTB Valid Delay	2	26	2	21	2	16	ns	
t44a	SEN Setup Time	11		9		7		ns	
t44b	SSTB # Setup Time	11		5		5		ns	
t45	SEN, SSTB # Setup Time	5		5		2		ns	
M/S # = V_{CC} (Master Mode)									
t46	BHOLD Setup Time	17		15		11		ns	
t47	BHOLD Hold Time	5		3		2		ns	
t48	BHLDA Valid Delay	5	28	4	23	3	16	ns	
M/S # = V_{SS} (Slave Mode)									
t49	BHLDA Setup Time	17		15		11		ns	
t50	BHLDA Hold Delay	5		3		2		ns	
t51	BHOLD Valid Delay	5	28	4	23	3	18	ns	
t55a	BLOCK # Valid Delay	4	30	4	26	3	20	ns	(Notes 1,5)
t55b1	BBE(0-3) # Valid Delay	4	30	4	26	3	20	ns	(Notes 1, 7)
t55b2	BBE(0-3 #) Valid Delay	4	30	4	26	3	20	ns	(Notes 1, 7)
t55b3	BBE(0-3) # Valid Delay	4	36	4	32	3	23	ns	(Notes 1, 7)
t55c	LOCK # Valid to BLOCK # Valid	0	30	0	26	0	20	ns	(Notes 1, 5)
t56	MISS # Valid Delay	4	35	4	30	3	22	ns	(Note 1)
t57	MISS #, BBE(0-3) #, BLOCK # Float Delay	4	32	4	30	4	25	ns	(Note 3)
t58	WBS Valid Delay	4	37	4	25	3	16	ns	(Note 1)
t59	FLUSH Setup Time	16		12		10		ns	
t60	FLUSH Hold Time	5		5		3		ns	
t61	FLUSH Setup to RESET Falling	26		21		16		ns	(Note 4)
t62	FLUSH Hold to RESET Falling	26		21		16		ns	(Note 4)

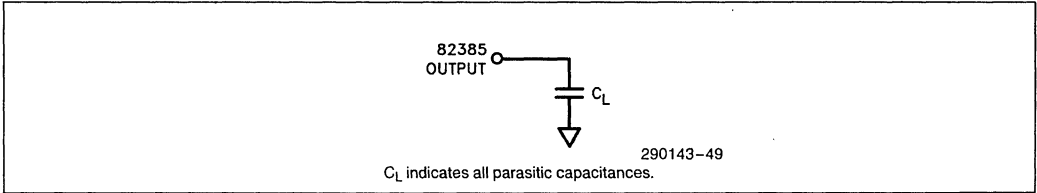
NOTES:

1. Frequency dependent specification.
2. Used for cache data memory (SRAM) specifications.
3. Float times not 100% tested.
4. This feature is tested only at 16 MHz.
5. BLOCK # delay is either from BPH11 or from 386 LOCK#. Refer to Figure 5-3K and 5-3L in the 82385 data sheet.
6. NCA # setup time is now specified to the rising edge of PHI2 in the state after 386 DX addresses become valid (either the first T2 or the state after the first T2P).
7. BBE # Valid delay is a function of NCA # setup.
8. Not 100% tested.
9. t5 is measured from 0.8V to 3.7V.
t6 is measured from 3.7V to 0.8V
This parameter is not 100% tested and is guaranteed by Intel's test methodology.



290143-48

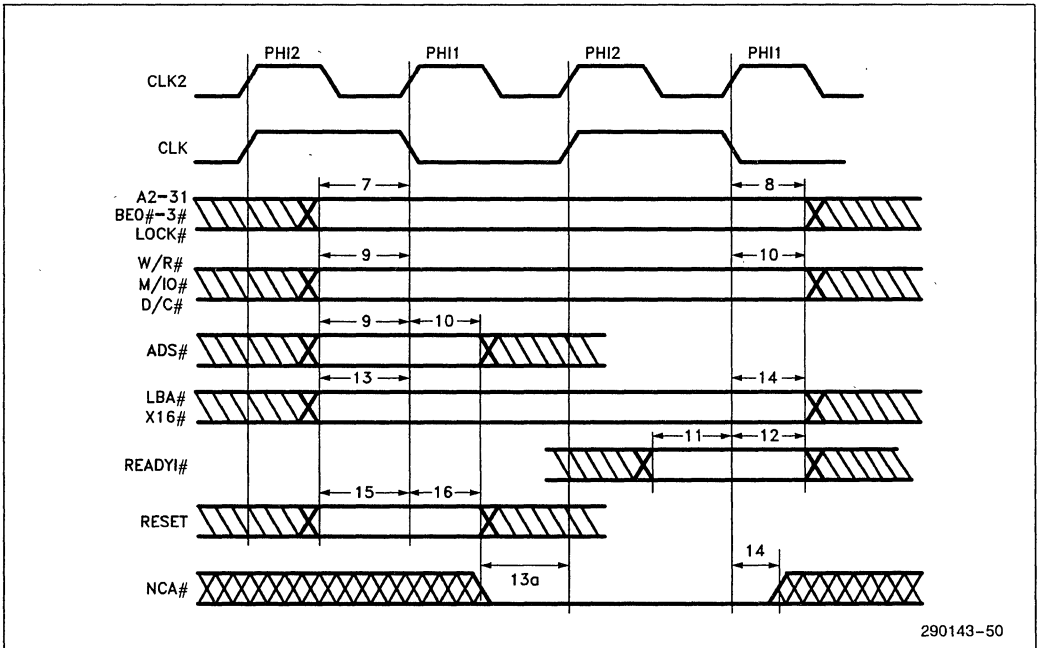
Figure 9-2. CLK2, BCLK2 Timing



290143-49

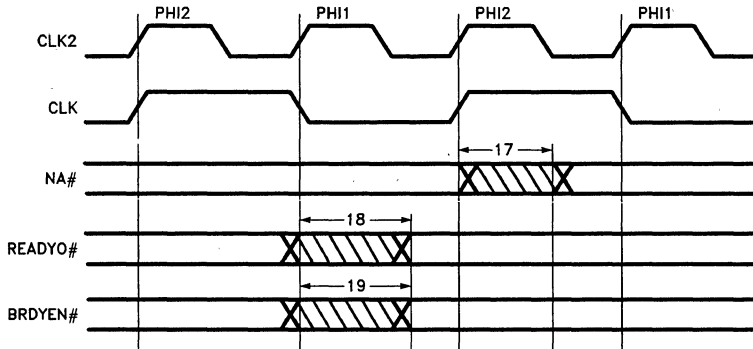
Figure 9-3. A.C. Test Load

386 DX Interface Parameters



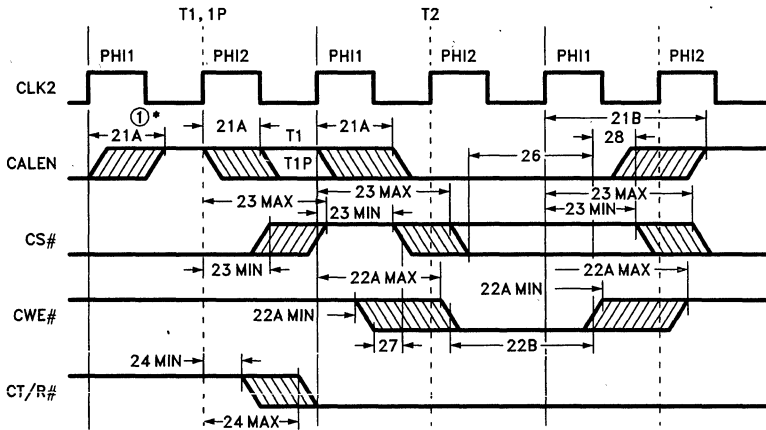
290143-50

OUTPUT DELAYS



290143-51

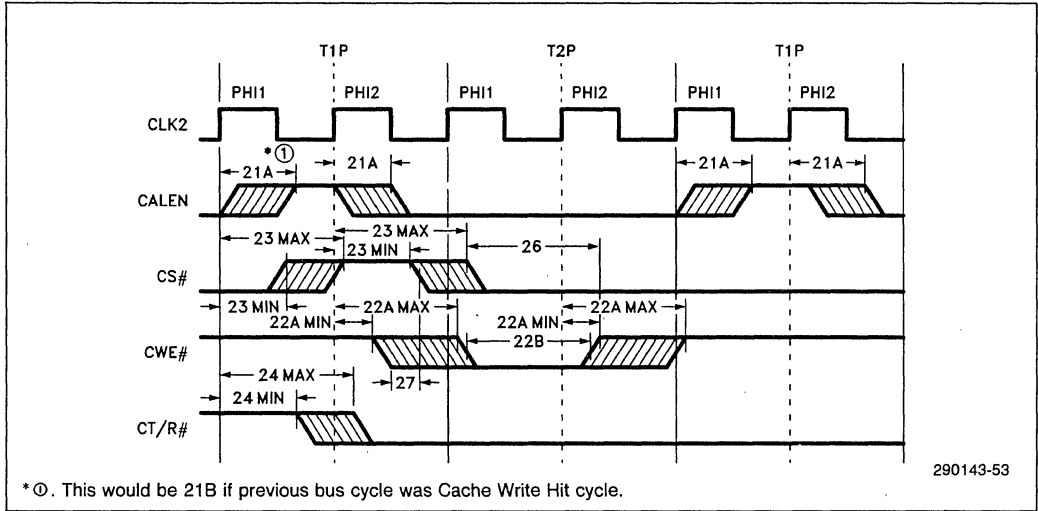
Cache Write Hit Cycle



⊙*. This would be 21B if previous bus cycle was Cache Write Hit cycle.

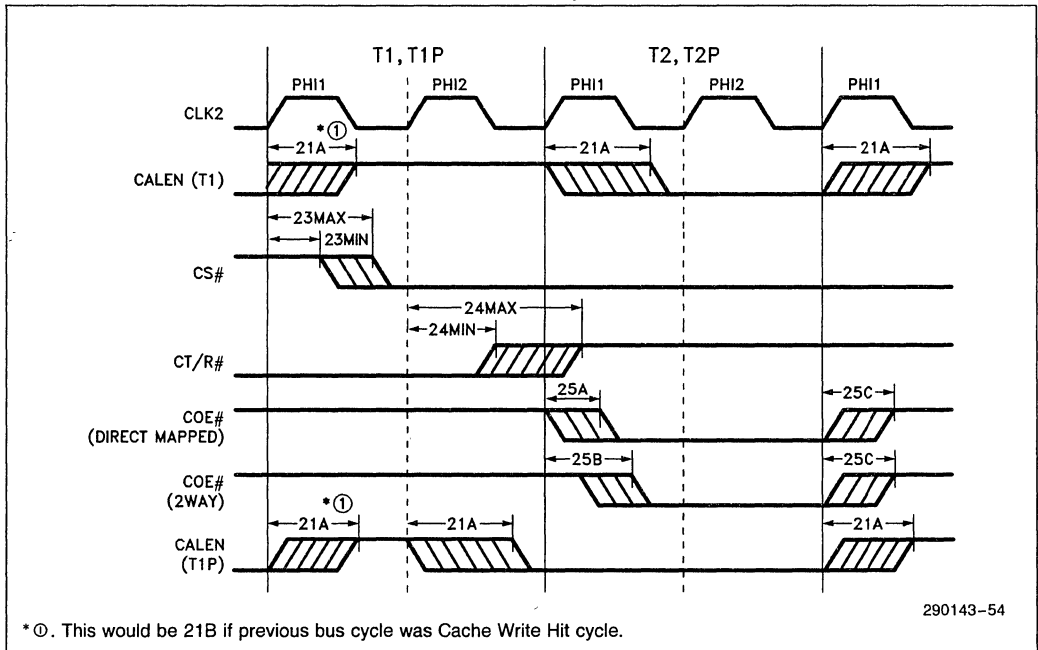
290143-52

Cache Read Miss (Cache Update Cycle)



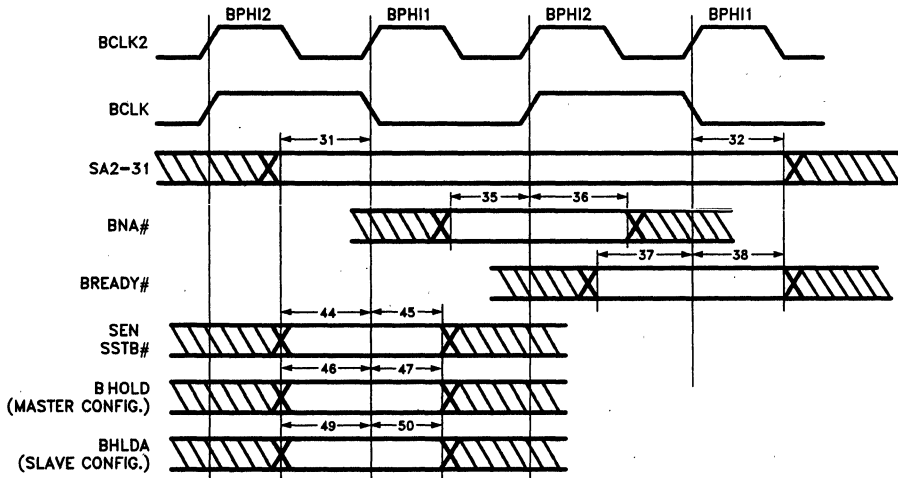
*⊙. This would be 21B if previous bus cycle was Cache Write Hit cycle.

Cache Read Cycle



*⊙. This would be 21B if previous bus cycle was Cache Write Hit cycle.

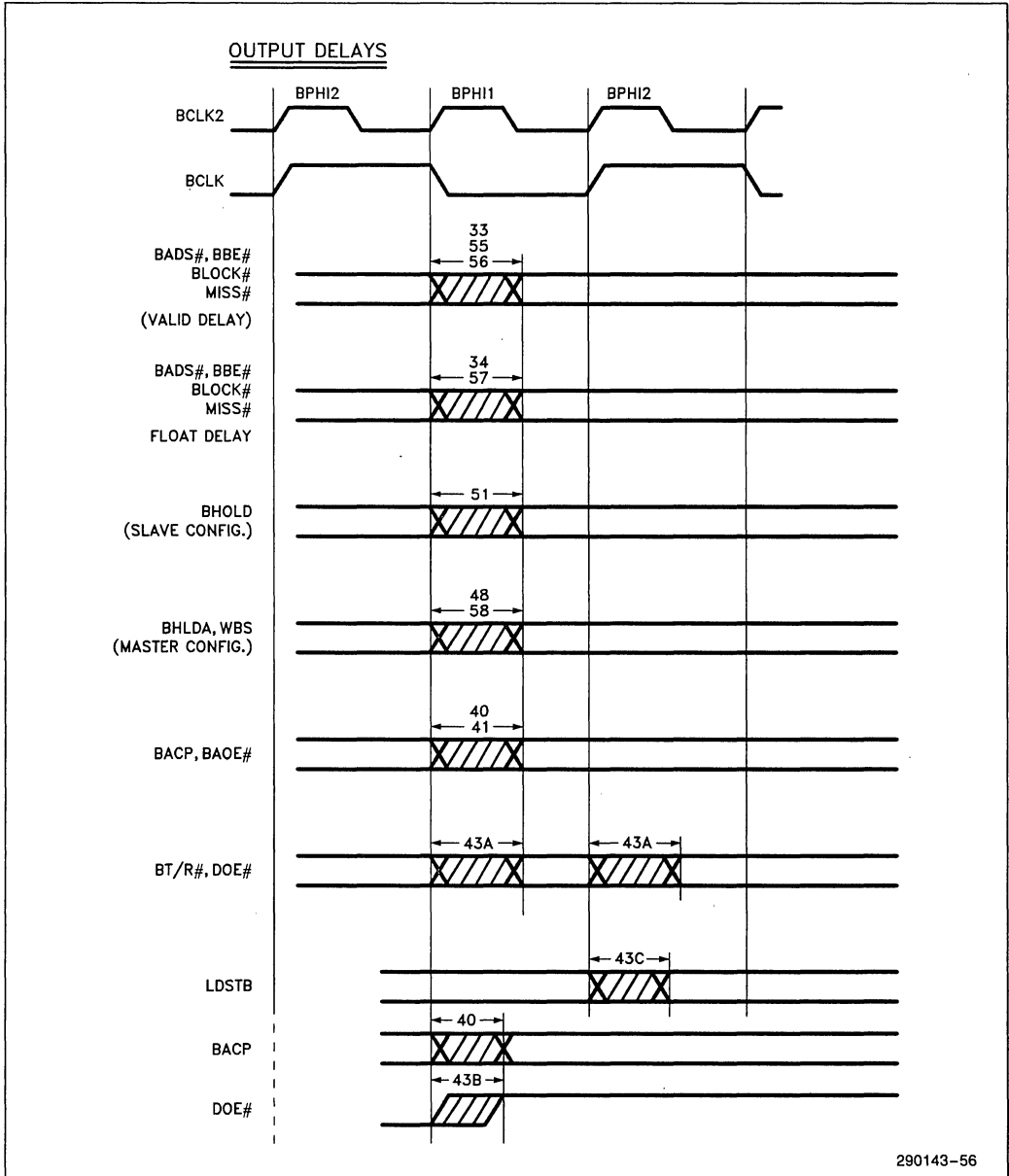
System Bus Interface Parameters



*This would be 21B if previous cycle was Cache Write Hit.

290143-55

System Bus Interface Parameters (Continued)



APPENDIX A

82385 Signal Summary

Signal Group/Name	Signal Function	Active State	Input/Output	Tri-State Output?
386 DX INTERFACE				
RESET	386 DX Reset	High	I	—
A2–A31	386 DX Address Bus	High	I	—
BE0#–BE3#	386 DX Byte Enables	Low	I	—
CLK2	386 DX Clock	—	I	—
READYO#	Ready Output	Low	O	No
BRDYEN#	Bus Ready Enable	Low	O	No
READYI#	386 DX Ready Input	Low	I	—
ADS#	386 DX Address Status	Low	I	—
M/IO#	386 DX Memory / I/O Indication	—	I	—
W/R#	386 DX Write/Read Indication	—	I	—
D/C#	386 DX Data/Control Indication	—	I	—
LOCK#	386 DX Lock Indication	Low	I	—
NA#	386 DX Next Address Request	Low	O	No
CACHE CONTROL				
CALEN	Cache Address Latch Enable	High	O	No
CT/R#	Cache Transmit/Receive	—	O	No
CS0#–CS3#	Cache Chip Selects	Low	O	No
COEA#, COEB#	Cache Output Enables	Low	O	No
CWEA#, CWEB#	Cache Write Enables	Low	O	No
LOCAL DECODE				
LBA#	386 DX Local Bus Access	Low	I	—
NCA#	Non-Cacheable Access	Low	I	—
X16#	16-Bit Access	Low	I	—
STATUS AND CONTROL				
MISS#	Cache Miss Indication	Low	O	Yes
WBS	Write Buffer Status	High	O	No
FLUSH	Cache Flush	High	I	—
82385 INTERFACE				
BREADY#	385 Ready Input	Low	I	—
BNA#	385 Next Address Request	Low	I	—
BLOCK#	385 Lock Indication	Low	O	Yes
BADS#	385 Address Status	Low	O	Yes
BBE0#–BBE3#	385 Byte Enables	Low	O	Yes

82385 Signal Summary (Continued)

Signal Group/Name	Signal Function	Active State	Input/Output	Tri-State Output?
DATA/ADDR CONTROL				
LDSTB	Local Data Strobe	Pos. Edge	0	No
DOE #	Data Output Enable	Low	0	No
BT/R #	Bus Transmit/Receive	—	0	No
BACP	Bus Address Clock Pulse	Pos. Edge	0	No
BAOE #	Bus Address Output Enable	Low	0	No
CONFIGURATION				
2W/D #	2-Way/Direct Map Select	—	1	—
M/S #	Master/Slave Select	—	1	—
DEFOE #	Define Cache Output Enable	—	1	—
COHERENCY				
SA2-SA31	Snoop Address Bus	High	1	—
SSTB #	Snoop Strobe	Low	1	—
SEN	Snoop Enable	High	1	—
ARBITRATION				
BHOLD	Hold	High	I/O	No
BHLDA	Hold Acknowledge	High	I/O	No

10.0 REVISION HISTORY

DOCUMENT: ADVANCE INFORMATION DATA SHEET			
PRIOR REV: 290143-003 September 1988			
NEW REV: 290143-004 September 1989			
Change #	Page #	Para. #	Change
1.	Throughout	Fig. 8-3	PQFP Package added
2.	Throughout	Tables 8-2, 8-3	PQFP Info
3.	Throughout	Table 8-4	PQFP Thermal Resistance
4.	Throughout		A.C. Specifications Unified (20 MHz, 25 MHz, 33 MHz)
5.	Throughout		DEFOE # Specifications added to device

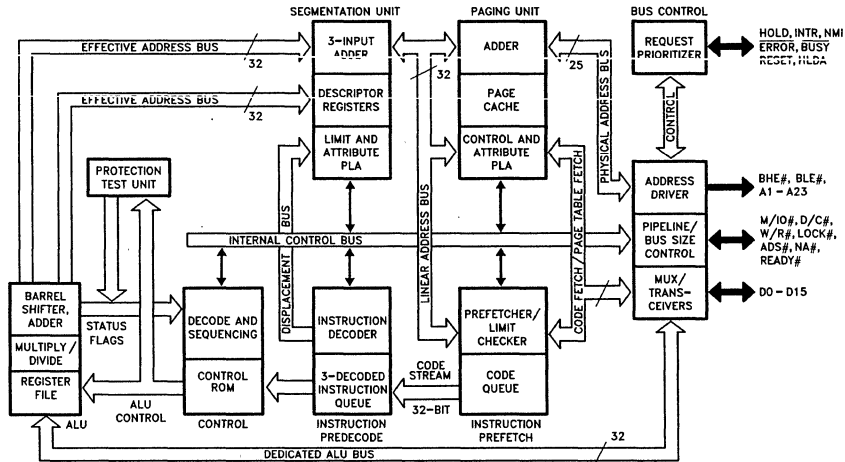


386™ SX MICROPROCESSOR

- Full 32-Bit Internal Architecture
 - 8-, 16-, 32-Bit Data Types
 - 8 General Purpose 32-Bit Registers
- Runs Intel386™ Software in a Cost Effective 16-Bit Hardware Environment
 - Runs Same Applications and O.S.'s as the 386™ DX Processor
 - Object Code Compatible with 8086, 80186, 80286, and 386 Processors
 - Runs MS-DOS*, OS/2* and UNIX**
- Very High Performance 16-Bit Data Bus
 - 20 MHz Clock
 - Two-Clock Bus Cycles
 - 20 Megabytes/Sec Bus Bandwidth
 - Address Pipelining Allows Use of Slower/Cheaper Memories
- Integrated Memory Management Unit
 - Virtual Memory Support
 - Optional On-Chip Paging
 - 4 Levels of Hardware Enforced Protection
 - MMU Fully Compatible with Those of the 80286 and 386 DX CPUs
- Virtual 8086 Mode Allows Execution of 8086 Software in a Protected and Paged System
- Large Uniform Address Space
 - 16 Megabyte Physical
 - 64 Terabyte Virtual
 - 4 Gigabyte Maximum Segment Size
- High Speed Numerics Support with the 387™ SX Coprocessor
- On-Chip Debugging Support Including Breakpoint Registers
- Complete System Development Support
 - Software: C, PL/M, Assembler
 - Debuggers: PMON-386 DX, ICETM-386 SX
 - Extensive Third-Party Support: C, Pascal, FORTRAN, BASIC, Ada*** on VAX, UNIX**, MS-DOS*, and Other Hosts
- High Speed CHMOS III Technology
- 100-Pin Plastic Quad Flatpack Package

(See Packaging Outlines and Dimensions # 231369)

The 386™ SX Microprocessor is a 32-bit CPU with a 16-bit external data bus and a 24-bit external address bus. The 386 SX CPU brings the high-performance software of the Intel386™ Architecture to midrange systems. It provides the performance benefits of a 32-bit programming architecture with the cost savings associated with 16-bit hardware systems.

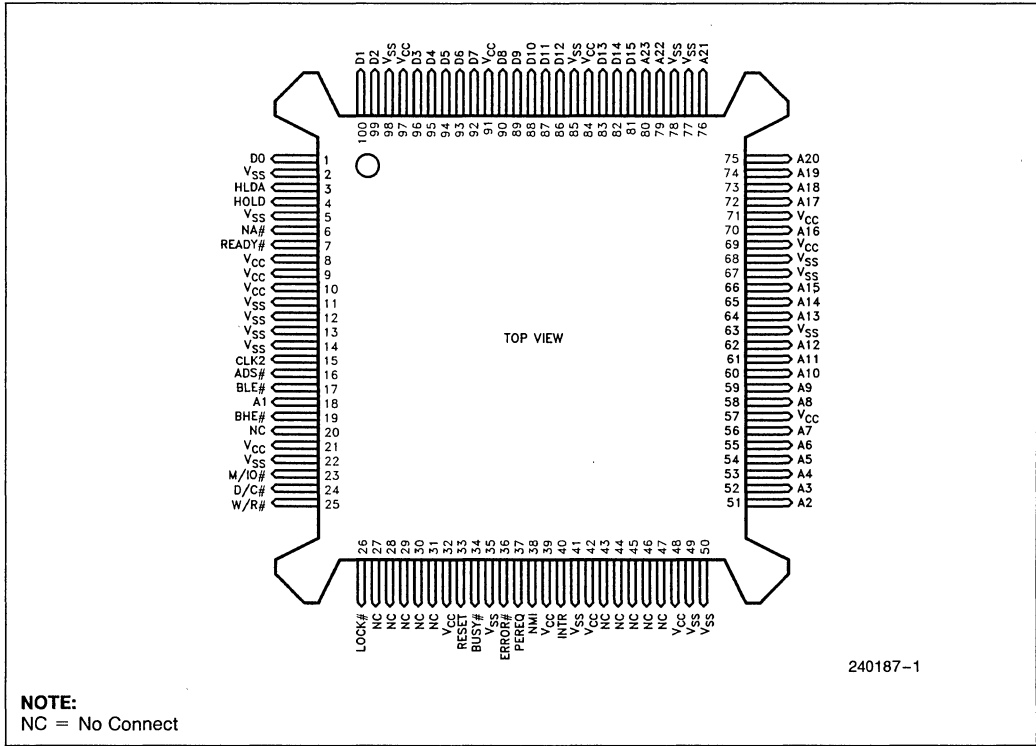


240187-47

386™ SX Pipelined 32-Bit Microarchitecture

*MS-DOS and OS/2 are trademarks of Microsoft Corporation.
 **UNIX is a trademark of AT&T.
 ***Ada is a trademark of the Department of Defense.

1.0 PIN DESCRIPTION



240187-1

Figure 1.1. 386™ SX Microprocessor Pin out Top View

Table 1.1. Alphabetical Pin Assignments

Address	Data	Control	N/C	Vcc	Vss
A ₁	D ₀	ADS#	16	20	2
A ₂	D ₁	BHE #	19	27	5
A ₃	D ₂	BLE #	17	28	11
A ₄	D ₃	BUSY #	34	29	12
A ₅	D ₄	CLK2	15	30	13
A ₆	D ₅	D/C#	24	31	14
A ₇	D ₆	ERROR #	36	43	22
A ₈	D ₇	HLDA	3	44	35
A ₉	D ₈	HOLD	4	45	41
A ₁₀	D ₉	INTR	40	46	49
A ₁₁	D ₁₀	LOCK#	26	47	50
A ₁₂	D ₁₁	M/IO#	23	84	63
A ₁₃	D ₁₂	NA#	6	91	67
A ₁₄	D ₁₃	NMI	38	97	68
A ₁₅	D ₁₄	PEREQ	37		77
A ₁₆	D ₁₅	READY #	7		78
A ₁₇		RESET	33		85
A ₁₈		W/R #	25		98
A ₁₉					
A ₂₀					
A ₂₁					
A ₂₂					
A ₂₃					

1.0 PIN DESCRIPTION (Continued)

The following are the 386™ SX Microprocessor pin descriptions. The following definitions are used in the pin descriptions:

- # The named signal is active LOW.
- I Input signal.
- O Output signal.
- I/O Input and Output signal.
- No electrical connection.

Symbol	Type	Pin	Name and Function
CLK2	I	15	CLK2 provides the fundamental timing for the 386™ SX Microprocessor. For additional information see Clock .
RESET	I	33	RESET suspends any operation in progress and places the 386™ SX Microprocessor in a known reset state. See Interrupt Signals for additional information.
D ₁₅ -D ₀	I/O	81-83,86-90, 92-96,99-100,1	Data Bus inputs data during memory, I/O and interrupt acknowledge read cycles and outputs data during memory and I/O write cycles. See Data Bus for additional information.
A ₂₃ -A ₁	O	80-79,76-72,70, 66-64,62-58, 56-51,18	Address Bus outputs physical memory or port I/O addresses. See Address Bus for additional information.
W/R#	O	25	Write/Read is a bus cycle definition pin that distinguishes write cycles from read cycles. See Bus Cycle Definition Signals for additional information.
D/C#	O	24	Data/Control is a bus cycle definition pin that distinguishes data cycles, either memory or I/O, from control cycles which are: interrupt acknowledge, halt, and code fetch. See Bus Cycle Definition Signals for additional information.
M/IO#	O	23	Memory/IO is a bus cycle definition pin that distinguishes memory cycles from input/output cycles. See Bus Cycle Definition Signals for additional information.
LOCK#	O	26	Bus Lock is a bus cycle definition pin that indicates that other system bus masters are not to gain control of the system bus while it is active. See Bus Cycle Definition Signals for additional information.
ADS#	O	16	Address Status indicates that a valid bus cycle definition and address (W/R#, D/C#, M/IO#, BHE#, BLE# and A ₂₃ -A ₁ are being driven at the 386™ SX Microprocessor pins. See Bus Control Signals for additional information.
NA#	I	6	Next Address is used to request address pipelining. See Bus Control Signals for additional information.
READY#	I	7	Bus Ready terminates the bus cycle. See Bus Control Signals for additional information.
BHE#, BLE#	O	19,17	Byte Enables indicate which data bytes of the data bus take part in a bus cycle. See Address Bus for additional information.

1.0 PIN DESCRIPTION (Continued)

Symbol	Type	Pin	Name and Function
HOLD	I	4	Bus Hold Request input allows another bus master to request control of the local bus. See Bus Arbitration Signals for additional information.
HLDA	O	3	Bus Hold Acknowledge output indicates that the 386™ SX Microprocessor has surrendered control of its local bus to another bus master. See Bus Arbitration Signals for additional information.
INTR	I	40	Interrupt Request is a maskable input that signals the 386™ SX Microprocessor to suspend execution of the current program and execute an interrupt acknowledge function. See Interrupt Signals for additional information.
NMI	I	38	Non-Maskable Interrupt Request is a non-maskable input that signals the 386™ SX Microprocessor to suspend execution of the current program and execute an interrupt acknowledge function. See Interrupt Signals for additional information.
BUSY#	I	34	Busy signals a busy condition from a processor extension. See Coprocessor Interface Signals for additional information.
ERROR#	I	36	Error signals an error condition from a processor extension. See Coprocessor Interface Signals for additional information.
PEREQ	I	37	Processor Extension Request indicates that the processor has data to be transferred by the 386™ SX Microprocessor. See Coprocessor Interface Signals for additional information.
N/C	-	20, 27-31, 43-47	No Connects should always be left unconnected. Connection of a N/C pin may cause the processor to malfunction or be incompatible with future steppings of the 386™ SX Microprocessor.
V _{cc}	I	8-10,21,32,39 42,48,57,69, 71,84,91,97	System Power provides the +5V nominal DC supply input.
V _{ss}	I	2,5,11-14,22 35,41,49-50, 63,67-68, 77-78,85,98	System Ground provides the 0V connection from which all inputs and outputs are measured.

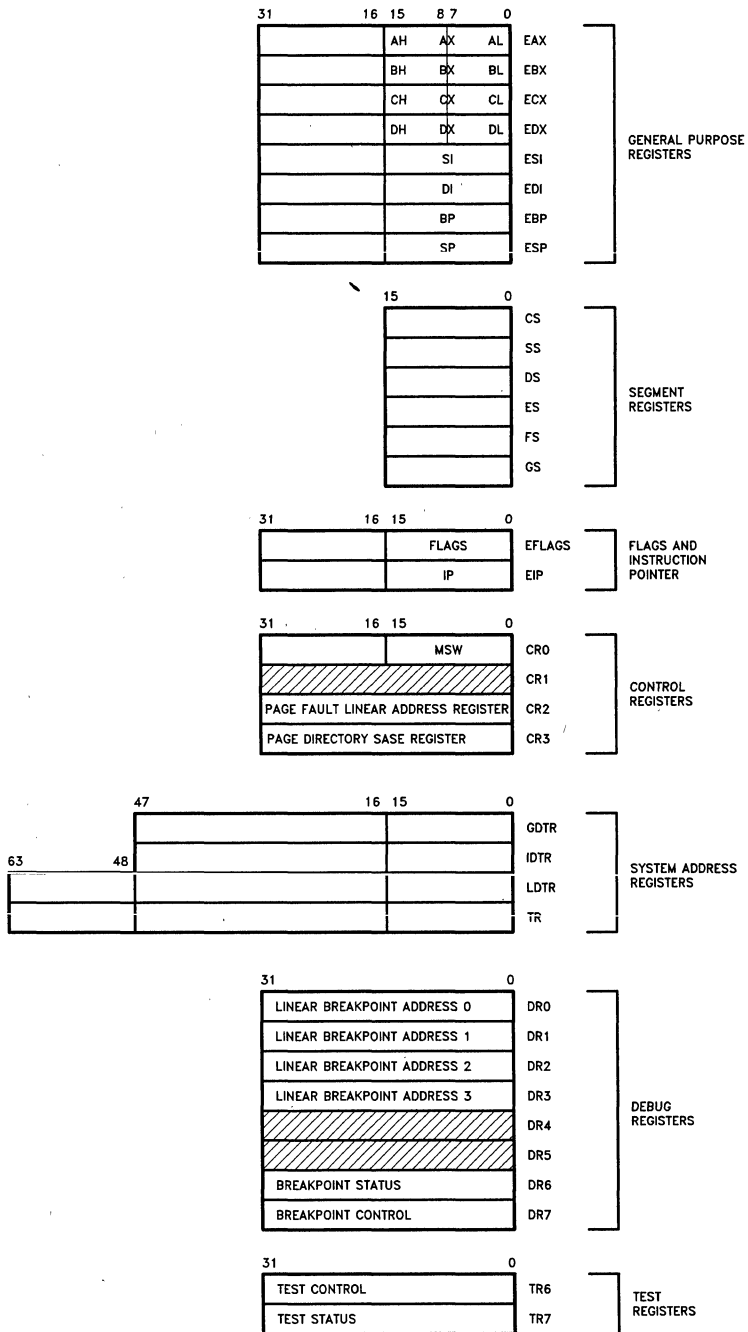


Figure 2.1. 386™ SX Microprocessor Registers

INTRODUCTION

The 386 SX Microprocessor is 100% object code compatible with the 386 DX, 286 and 8086 microprocessors. System manufacturers can provide 386 DX CPU based systems optimized for performance and 386 SX CPU based systems optimized for cost, both sharing the same operating systems and application software. Systems based on the 386 SX CPU can access the world's largest existing microcomputer software base, including the growing 32-bit software base. Only the Intel386 DX architecture can run UNIX, OS/2 and MS-DOS.

Instruction pipelining, high bus bandwidth, and a very high performance ALU ensure short average instruction execution times and high system throughput. The 386 SX CPU is capable of execution at sustained rates of 2.5–3.0 million instructions per second.

The integrated memory management unit (MMU) includes an address translation cache, advanced multi-tasking hardware, and a four-level hardware-enforced protection mechanism to support operating systems. The virtual machine capability of the 386 SX CPU allows simultaneous execution of applications from multiple operating systems such as MS-DOS and UNIX.

The 386 SX CPU offers on-chip testability and debugging features. Four breakpoint registers allow conditional or unconditional breakpoint traps on code execution or data accesses for powerful debugging of even ROM-based systems. Other testability features include self-test, tri-state of output buffers, and direct access to the page translation cache.

2.0 BASE ARCHITECTURE

The 386 SX Microprocessor consists of a central processing unit, a memory management unit and a bus interface.

The central processing unit consists of the execution unit and the instruction unit. The execution unit contains the eight 32-bit general purpose registers which are used for both address calculation and data operations and a 64-bit barrel shifter used to speed shift, rotate, multiply, and divide operations. The instruction unit decodes the instruction opcodes and stores them in the decoded instruction queue for immediate use by the execution unit.

The memory management unit (MMU) consists of a segmentation unit and a paging unit. Segmentation allows the managing of the logical address space by

providing an extra addressing component, one that allows easy code and data relocatability, and efficient sharing. The paging mechanism operates beneath and is transparent to the segmentation process, to allow management of the physical address space.

The segmentation unit provides four levels of protection for isolating and protecting applications and the operating system from each other. The hardware enforced protection allows the design of systems with a high degree of integrity.

The 386 SX Microprocessor has two modes of operation: Real Address Mode (Real Mode), and Protected Virtual Address Mode (Protected Mode). In Real Mode the 386 SX Microprocessor operates as a very fast 8086, but with 32-bit extensions if desired. Real Mode is required primarily to set up the processor for Protected Mode operation.

Within Protected Mode, software can perform a task switch to enter into tasks designated as Virtual 8086 Mode tasks. Each such task behaves with 8086 semantics, thus allowing 8086 software (an application program or an entire operating system) to execute. The Virtual 8086 tasks can be isolated and protected from one another and the host 386 SX Microprocessor operating system by use of paging.

Finally, to facilitate high performance system hardware designs, the 386 SX Microprocessor bus interface offers address pipelining and direct Byte Enable signals for each byte of the data bus.

2.1 Register Set

The 386 SX Microprocessor has thirty-four registers as shown in Figure 2-1. These registers are grouped into the following seven categories:

General Purpose Registers: The eight 32-bit general purpose registers are used to contain arithmetic and logical operands. Four of these (EAX, EBX, ECX, and EDX) can be used either in their entirety as 32-bit registers, as 16-bit registers, or split into pairs of separate 8-bit registers.

Segment Registers: Six 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data.

Flags and Instruction Pointer Registers: The two 32-bit special purpose registers in figure 2.1 record or control certain aspects of the 386 SX Microprocessor state. The EFLAGS register includes status and control bits that are used to reflect the outcome of many instructions and modify the semantics of

some instructions. The Instruction Pointer, called EIP, is 32 bits wide. The Instruction Pointer controls instruction fetching and the processor automatically increments it after executing an instruction.

Control Registers: The four 32-bit control register are used to control the global nature of the 386 SX Microprocessor. The CR0 register contains bits that set the different processor modes (Protected, Real, Paging and Coprocessor Emulation). CR2 and CR3 registers are used in the paging operation.

System Address Registers: These four special registers reference the tables or segments supported by the 80286/386 SX/386 DX CPU's protection model. These tables or segments are:

- GDTR (Global Descriptor Table Register),
- IDTR (Interrupt Descriptor Table Register),
- LDTR (Local Descriptor Table Register),
- TR (Task State Segment Register).

Debug Registers: The six programmer accessible debug registers provide on-chip support for debugging. The use of the debug registers is described in Section 2.10 **Debugging Support**.

Test Registers: Two registers are used to control the testing of the RAM/CAM (Content Addressable Memories) in the Translation Lookaside Buffer portion of the 386 SX Microprocessor. Their use is discussed in **Testability**.

EFLAGS REGISTER

The flag register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS, shown in Figure 2.2, control certain operations and indicate the status of the 386 SX Microprocessor. The lower 16 bits (bits 0–15) of EFLAGS contain the 16-bit flag register named FLAGS. This is the default flag register used when executing 8086, 80286, or real mode code. The functions of the flag bits are given in Table 2.1.

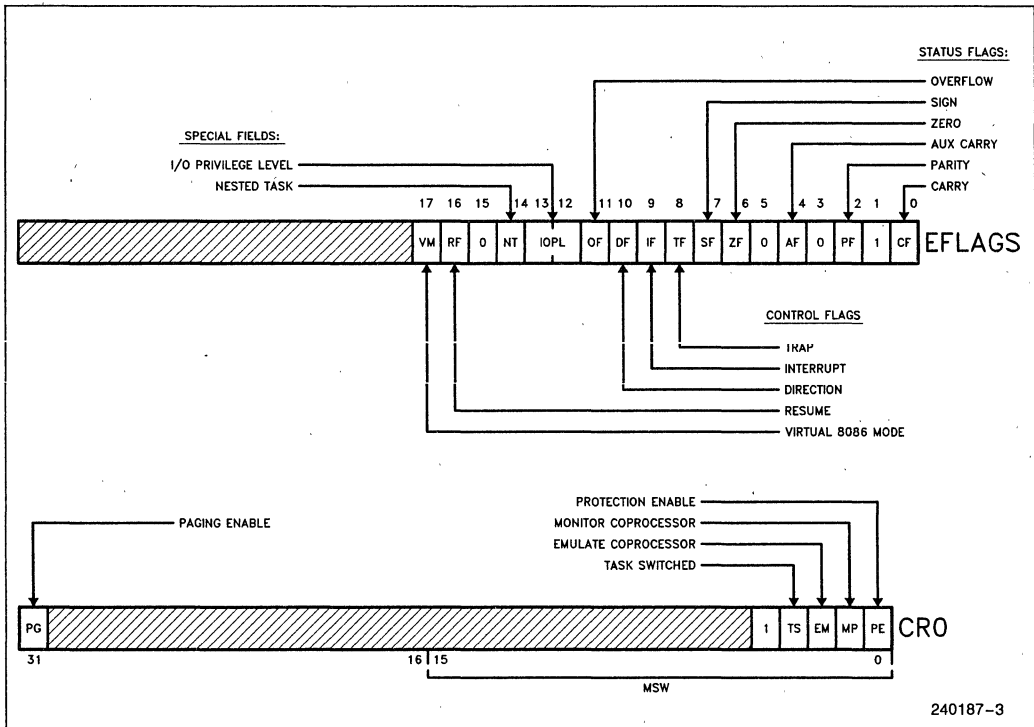


Figure 2.2. Status and Control Register Bit Functions

Table 2.1. Flag Definitions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise.
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise.
4	AF	Auxiliary Carry Flag—Set on carry from or borrow to the low order four bits of AL; cleared otherwise.
6	ZF	Zero Flag—Set if result is zero; cleared otherwise.
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative).
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-Enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto-increment (default) the appropriate index registers when cleared. Setting DF causes auto-decrement.
11	OF	Overflow Flag—Set if the operation resulted in a carry/borrow into the sign bit (high-order bit) of the result but did not result in a carry/borrow out of the high-order bit or vice-versa.
12,13	IOPL	I/O Privilege Level—Indicates the maximum CPL permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O permission bit map while executing in protected mode. For virtual 86 mode it indicates the maximum CPL allowing alteration of the IF bit.
14	NT	Nested Task—Indicates that the execution of the current task is nested within another task.
16	RF	Resume Flag—Used in conjunction with debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. If set, any debug fault is ignored on the next instruction.
17	VM	Virtual 8086 Mode—If set while in protected mode, the 386™ SX Microprocessor will switch to virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes.

CONTROL REGISTERS

The 386 SX Microprocessor has three control registers of 32 bits, CR0, CR2 and CR3, to hold the machine state of a global nature. These registers are shown in Figures 2.1 and 2.2. The defined CR0 bits are described in Table 2.2.

Table 2.2. CR0 Definitions

Bit Position	Name	Function
0	PE	Protection mode enable—places the 386™ SX Microprocessor into protected mode. If PE is reset, the processor operates again in Real Mode. PE may be set by loading MSW or CR0. PE can be reset only by loading CR0, it cannot be reset by the LMSW instruction.
1	MP	Monitor coprocessor extension—allows WAIT instructions to cause a processor extension not present exception (number 7).
2	EM	Emulate processor extension—causes a processor extension not present exception (number 7) on ESC instructions to allow emulating a processor extension.
3	TS	Task switched—indicates the next instruction using a processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task.
31	PG	Paging enable bit—is set to enable the on-chip paging unit. It is reset to disable the on-chip paging unit.

2.2 Instruction Set

The instruction set is divided into nine categories of operations:

- Data Transfer
- Arithmetic
- Shift/Rotate
- String Manipulation
- Bit Manipulation
- Control Transfer
- High Level Language Support
- Operating System Support
- Processor Control

These instructions are listed in Table 9.1 **Instruction Set Clock Count Summary**.

All 386 SX Microprocessor instructions operate on either 0, 1, 2 or 3 operands; an operand resides in a register, in the instruction itself, or in memory. Most zero operand instructions (e.g CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the 386 SX Microprocessor has a 16 byte prefetch instruction queue, an average of 5 instructions will be prefetched. The use of two operands permits the following types of common instructions:

- Register to Register
- Memory to Register
- Immediate to Register
- Memory to Memory
- Register to Memory
- Immediate to Memory.

The operands can be either 8, 16, or 32 bits long. As a general rule, when executing code written for the 386 SX Microprocessor (32 bit code), operands are 8 or 32 bits; when executing existing 8086 or 80286 code (16-bit code), operands are 8 or 16 bits. Prefixes can be added to all instructions which override the default length of the operands (i.e. use 32-bit operands for 16-bit code, or 16-bit operands for 32-bit code).

2.3 Memory Organization

Memory on the 386 SX Microprocessor is divided into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address. The address of a word or dword is the byte address of the low-order byte.

In addition to these basic data types, the 386 SX Microprocessor supports two larger units of memory: pages and segments. Memory can be divided up into one or more variable length segments, which can be swapped to disk or shared between programs. Memory can also be organized into one or more 4K byte pages. Finally, both segmentation and paging can be combined, gaining the advantages of both systems. The 386 SX Microprocessor supports both pages and segmentation in order to provide maximum flexibility to the system designer. Segmentation and paging are complementary. Segmentation is useful for organizing memory in logical modules, and as such is a tool for the application programmer, while pages are useful to the system programmer for managing the physical memory of a system.

ADDRESS SPACES

The 386 SX Microprocessor has three types of address spaces: **logical**, **linear**, and **physical**. A **logical address** (also known as a **virtual address**) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, DISPLACEMENT), discussed in section 2.4 **Addressing Modes**, into an effective address. This effective address along with the selector is known as the logical address. Since each task on the 386 SX Microprocessor has a maximum of 16K ($2^{14} - 1$) selectors, and offsets can be 4 gigabytes (with paging enabled) this gives a total of 2^{46} bits, or 64 terabytes, of **logical address space** per task. The programmer sees the logical address space.

The segmentation unit translates the **logical address space** into a 32-bit **linear address space**. If the paging unit is not enabled then the 32-bit **linear address** is truncated into a 24-bit **physical address**. The **physical address** is what appears on the address pins.

The primary differences between Real Mode and Protected Mode are how the segmentation unit performs the translation of the **logical address** into the **linear address**, size of the address space, and paging capability. In Real Mode, the segmentation unit shifts the selector left four bits and adds the result to the effective address to form the **linear address**. This **linear address** is limited to 1 megabyte. In addition, real mode has no paging capability.

Protected Mode will see one of two different address spaces, depending on whether or not paging is enabled. Every selector has a **logical base address** associated with it that can be up to 32 bits in length. This 32-bit **logical base address** is added to the effective address to form a final 32-bit **linear**

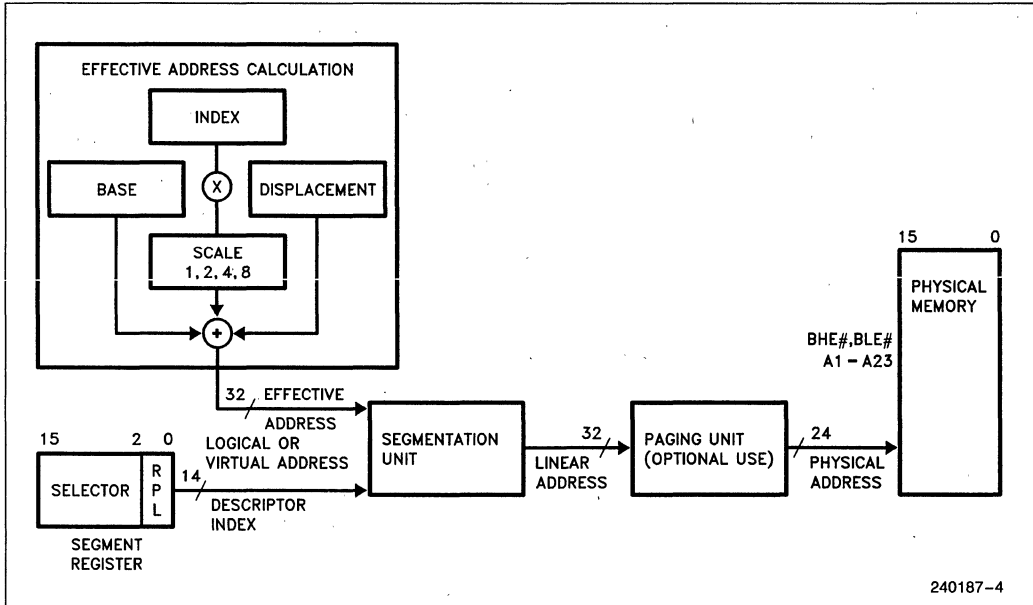


Figure 2.3. Address Translation

address. If paging is disabled this final **linear** address reflects physical memory and is truncated so that only the lower 24 bits of this address are used to address the 16 megabyte memory address space. If paging is enabled this final **linear** address reflects a 32-bit address that is translated through the paging unit to form a 16-megabyte physical address. The **logical base** address is stored in one of two operating system tables (i.e. the Local Descriptor Table or Global Descriptor Table).

Figure 2.3 shows the relationship between the various address spaces.

SEGMENT REGISTER USAGE

The main data structure used to organize memory is the segment. On the 386 SX Microprocessor, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments, code and data. The segments are of variable size and can be as small as 1 byte or as large as 4 gigabytes (2³² bits).

In order to provide compact instruction encoding and increase processor performance, instructions do not need to explicitly specify which segment register is used. The segment register is automatically chosen according to the rules of Table 2.3 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register, stack references use the SS register and instruction

fetches use the CS register. The contents of the Instruction Pointer provide the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2.3. The override prefixes also allow the use of the ES, FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero and create a system with a four gigabyte linear address space. This creates a system where the virtual address space is the same as the linear address space. Further details of segmentation are discussed in chapter 4 **PROTECTED MODE ARCHITECTURE**.

2.4 Addressing Modes

The 386 SX Microprocessor provides a total of 8 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

REGISTER AND IMMEDIATE MODES

Two of the addressing modes provide for instructions that operate on register or immediate operands:

Table 2.3. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA Instructions	SS	None
Source of POP, POPA, POPF, IRET, RET Instructions	SS	None
Destination of STOS, MOVE, REP STOS, and REP MOVS instructions	ES	None
Other data references, with effective address using base register of:		
[EAX]	DS	CS,SS,ES,FS,GS
[EBX]	DS	CS,SS,ES,FS,GS
[ECX]	DS	CS,SS,ES,FS,GS
[EDX]	DS	CS,SS,ES,FS,GS
[ESI]	DS	CS,SS,ES,FS,GS
[EDI]	DS	CS,SS,ES,FS,GS
[EBP]	SS	CS,DS,ES,FS,GS
[ESP]	SS	CS,DS,ES,FS,GS

Register Operand Mode: The operand is located in one of the 8, 16 or 32-bit general registers.

Immediate Operand Mode: The operand is included in the instruction as part of the opcode.

32-BIT MEMORY ADDRESSING MODES

The remaining 6 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the segment base address and an effective address. The effective address is calculated by summing any combination of the following three address elements (see figure 2.3):

DISPLACEMENT: an 8, 16 or 32-bit immediate value, following the instruction.

BASE: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area.

INDEX: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters. The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. The scaled index is especially useful for accessing arrays or structures.

Combinations of these 3 components make up the 6 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of Base and Index components which requires one additional clock.

As shown in Figure 2.4, the effective address (EA) of an operand is calculated according to the following formula:

$$EA = Base_{Register} + (Index_{Register} * scaling) + Displacement$$

- 1. Direct Mode:** The operand's offset is contained as part of the instruction as an 8, 16 or 32-bit displacement.
- 2. Register Indirect Mode:** A BASE register contains the address of the operand.
- 3. Based Mode:** A BASE register's contents are added to a DISPLACEMENT to form the operand's offset.
- 4. Scaled Index Mode:** An INDEX register's contents are multiplied by a SCALING factor, and the result is added to a DISPLACEMENT to form the operand's offset.

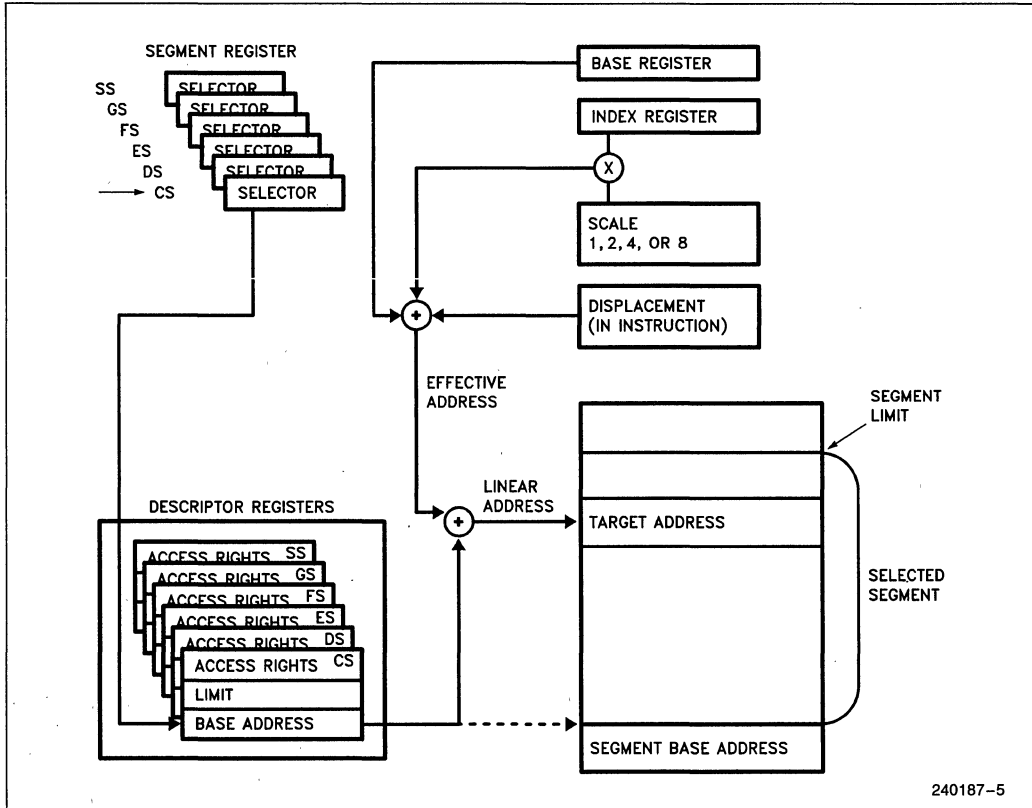


Figure 2.4. Addressing Mode Calculations

5. **Based Scaled Index Mode:** The contents of an INDEX register are multiplied by a SCALING factor, and the result is added to the contents of a BASE register to obtain the operand's offset.
6. **Based Scaled Index Mode with Displacement:** The contents of an INDEX register are multiplied by a SCALING factor, and the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.

DIFFERENCES BETWEEN 16 AND 32 BIT ADDRESSES

In order to provide software compatibility with the 8086 and the 80286, the 386 SX Microprocessor can execute 16-bit instructions in Real and Protected Modes. The processor determines the size of the instructions it is executing by examining the D bit in a Segment Descriptor. If the D bit is 0 then all operand lengths and effective addresses are assumed to be 16 bits long. If the D bit is 1 then the default length for operands and addresses is 32 bits. In Real Mode the default size for operands and addresses is 16 bits.

Regardless of the default precision of the operands or addresses, the 386 SX Microprocessor is able to execute either 16 or 32-bit instructions. This is specified through the use of override prefixes. Two prefixes, the **Operand Length Prefix** and the **Address Length Prefix**, override the value of the D bit on an individual instruction basis. These prefixes are automatically added by assemblers.

The Operand Length and Address Length Prefixes can be applied separately or in combination to any instruction. The Address Length Prefix does not allow addresses over 64K bytes to be accessed in Real Mode. A memory address which exceeds 0FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional 386 SX Microprocessor addressing modes.

When executing 32-bit code, the 386 SX Microprocessor uses either 8 or 32-bit displacements, and any register can be used as base or index registers. When executing 16-bit code, the displacements are either 8 or 16-bits, and the base and index register conform to the 80286 model. Table 2.4 illustrates the differences.

Table 2.4. BASE and INDEX Registers for 16- and 32-Bit Addresses

	16-Bit Addressing	32-Bit Addressing
BASE REGISTER	BX,BP	Any 32-bit GP Register
INDEX REGISTER	SI,DI	Any 32-bit GP Register
SCALE FACTOR	None	Except ESP
DISPLACEMENT	0, 8, 16-bits	1, 2, 4, 8 0, 8, 32-bits

2.5 Data Types

The 386 SX Microprocessor supports all of the data types commonly used in high level languages:

Bit: A single bit quantity.

Bit Field: A group of up to 32 contiguous bits, which spans a maximum of four bytes.

Bit String: A set of contiguous bits; on the 386 SX Microprocessor, bit strings can be up to 4 gigabits long.

Byte: A signed 8-bit quantity.

Unsigned Byte: An unsigned 8-bit quantity.

Integer (Word): A signed 16-bit quantity.

Long Integer (Double Word): A signed 32-bit quantity. All operations assume a 2's complement representation.

Unsigned Integer (Word): An unsigned 16-bit quantity.

Unsigned Long Integer (Double Word): An unsigned 32-bit quantity.

Signed Quad Word: A signed 64-bit quantity.

Unsigned Quad Word: An unsigned 64-bit quantity.

Pointer: A 16 or 32-bit offset-only quantity which indirectly references another memory location.

Long Pointer: A full pointer which consists of a 16-bit segment selector and either a 16 or 32-bit offset.

Char: A byte representation of an ASCII Alphanumeric or control character.

String: A contiguous sequence of bytes, words or dwords. A string may contain between 1 byte and 4 gigabytes

BCD: A byte (unpacked) representation of decimal digits 0–9.

Packed BCD: A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble.

When the 386 SX Microprocessor is coupled with its numerics coprocessor, the 387™ SX, then the following common floating point types are supported:

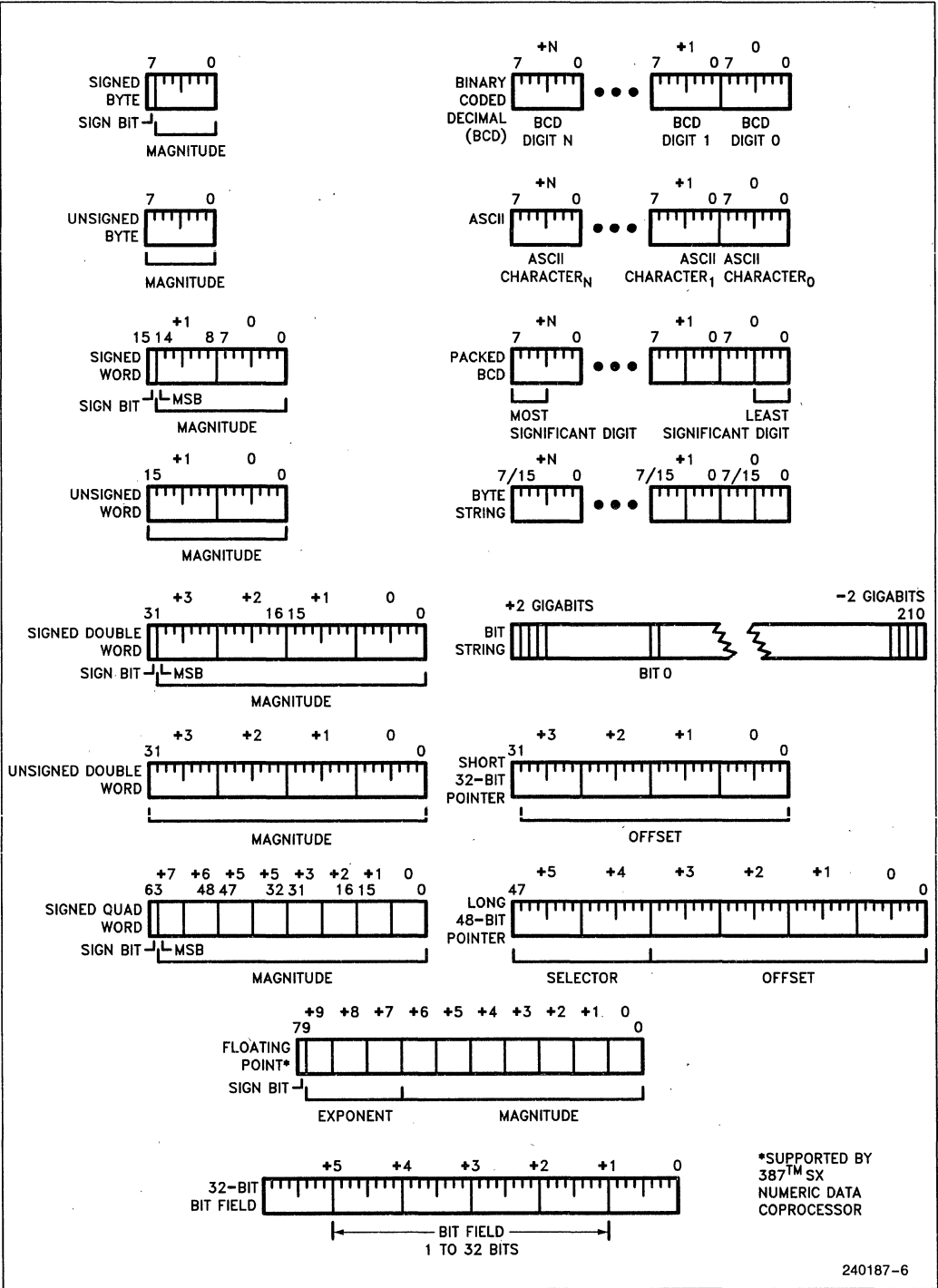
Floating Point: A signed 32, 64, or 80-bit real number representation. Floating point numbers are supported by the 387™ SX numerics coprocessor.

Figure 2.5 illustrates the data types supported by the 386 SX Microprocessor and the 387 SX.

2.6 I/O Space

The 386 SX Microprocessor has two distinct physical address spaces: physical memory and I/O. Generally, peripherals are placed in I/O space although the 386 SX Microprocessor also supports memory-mapped peripherals. The I/O space consists of 64K bytes which can be divided into 64K 8-bit ports or 32K 16-bit ports, or any combination of ports which add up to no more than 64K bytes. The 64K I/O address space refers to physical addresses rather than linear addresses since I/O instructions do not go through the segmentation or paging hardware. The M/IO# pin acts as an additional address line, thus allowing the system designer to easily determine which address space the processor is accessing.

The I/O ports are accessed by the IN and OUT instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8-bit and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/IO# pin to be driven LOW. I/O port addresses 00F8H through 00FFH are reserved for use by Intel.



*SUPPORTED BY 387™ SX NUMERIC DATA COPROCESSOR

Figure 2.5. 386™ SX Microprocessor Supported Data Types

Table 2.5. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	YES	FAULT
Debug Exception	1	any instruction	YES	TRAP*
NMI Interrupt	2	INT 2 or NMI	NO	NMI
One Byte Interrupt	3	INT	NO	TRAP
Interrupt on Overflow	4	INTO	NO	TRAP
Array Bounds Check	5	BOUND	YES	FAULT
Invalid OP-Code	6	Any illegal instruction	YES	FAULT
Device Not Available	7	ESC, WAIT	YES	FAULT
Double Fault	8	Any instruction that can generate an exception		ABORT
Coprocessor Segment Overrun	9	ESC	NO	ABORT
Invalid TSS	10	JMP, CALL, IRET, INT	YES	FAULT
Segment Not Present	11	Segment Register Instructions	YES	FAULT
Stack Fault	12	Stack References	YES	FAULT
General Protection Fault	13	Any Memory Reference	YES	FAULT
Page Fault	14	Any Memory Access or Code Fetch	YES	FAULT
Coprocessor Error	16	ESC, WAIT	YES	FAULT
Intel Reserved	17–32			
Two Byte Interrupt	0–255	INT n	NO	TRAP

*Some debug exceptions may report both traps on the previous instruction and faults on the next instruction.

2.7 Interrupts and Exceptions

Interrupts and exceptions alter the normal program flow in order to handle external events, report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT N instruction, the processor treats software interrupts as exceptions.

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately **after** the interrupted instruction.

Exceptions are classified as faults, traps, or aborts, depending on the way they are reported and whether or not restart of the instruction causing the exception is supported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined.

Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point to the instruction causing the exception and will include any leading instruction prefixes. Table 2.5 summarizes the possible interrupts for the 386 SX Microprocessor and shows where the return address points to.

The 386 SX Microprocessor has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode, the vectors are 4-byte quantities, a Code Segment plus a 16-bit offset; in Protected Mode, the interrupt vectors are 8 byte quantities, which are put in an Interrupt Descriptor Table. Of the 256 possible interrupts, 32 are reserved for use by Intel and the remaining 224 are free to be used by the system designer.

INTERRUPT PROCESSING

When an interrupt occurs, the following actions happen. First, the current program address and Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the 386 SX Microprocessor which identifies the appropriate entry in the interrupt table. The table contains the starting address of the interrupt service routine. Then, the user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the 386 SX Microprocessor in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

Maskable Interrupt

Maskable interrupts are the most common way to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled HIGH and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions (string instructions have an 'interrupt window' between memory moves which allows interrupts during long string moves). When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt (one of 224 user defined interrupts).

Interrupts through interrupt gates automatically reset IF, disabling INTR requests. Interrupts through Trap Gates leave the state of the IF bit unchanged. Interrupts through a Task Gate change the IF bit according to the image of the EFLAGS register in the task's Task State Segment (TSS). When an IRET instruction is executed, the original state of the IF bit is restored.

Non-Maskable Interrupt

Non-maskable interrupts provide a method of servicing very high priority interrupts. When the NMI input is pulled HIGH it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt, no interrupt acknowledgment sequence is performed for an NMI.

While executing the NMI servicing procedure, the 386 SX Microprocessor will not service any further NMI request or INT requests until an interrupt return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The IF bit is cleared at the beginning of an NMI interrupt to inhibit further INTR interrupts.

Software Interrupts

A third type of interrupt/exception for the 386 SX Microprocessor is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table.

A special case of the two byte software interrupt INT n is the one byte INT 3, or breakpoint interrupt. By inserting this one byte instruction in a program, the user can set breakpoints in his program as a debugging tool.

A final type of software interrupt is the single step interrupt. It is discussed in **Single Step Trap**.

INTERRUPT AND EXCEPTION PRIORITIES

Interrupts are externally generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are **both** recognized at the **same** instruction boundary, the 386 SX Microprocessor invokes the NMI service routine first. If maskable interrupts are still enabled after the NMI service routine has been invoked, then the 386 SX Microprocessor will invoke the appropriate interrupt service routine.

As the 386 SX Microprocessor executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2.6. This cycle is re-

peated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

INSTRUCTION RESTART

The 386 SX Microprocessor fully supports restarting all instructions after Faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10 in Table 2.6), the 386 SX Microprocessor invokes the appropriate exception service routine. The 386 SX Microprocessor is in a state that permits restart of the instruction, for all cases but those given in Table 2.7. Note that all such cases will be avoided by a properly designed operating system.

Table 2.6. Sequence of Exception Checking

Consider the case of the 386™ SX Microprocessor having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for external NMI and INTR.
3. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Page Faults that prevented fetching the entire next instruction (exception 14).
6. Check for Faults decoding the next instruction (exception 6 if illegal opcode; exception 6 if in Real Mode or in Virtual 8086 Mode and attempting to execute an instruction for Protected Mode only; or exception 13 if instruction is longer than 15 bytes, or privilege violation in Protected Mode (i.e. not at IOPL or at CPL=0).
7. If WAIT opcode, check if TS=1 and MP=1 (exception 7 if both are 1).
8. If ESCape opcode for numeric coprocessor, check if EM=1 or TS=1 (exception 7 if either are 1).
9. If WAIT opcode or ESCape opcode for numeric coprocessor, check ERROR# input signal (exception 16 if ERROR# input is asserted).
10. Check in the following order for each memory reference required by the instruction:
 - a. Check for Segmentation Faults that prevent transferring the entire memory quantity (exceptions 11, 12, 13).
 - b. Check for Page Faults that prevent transferring the entire memory quantity (exception 14).

NOTE:

Segmentation exceptions are generated before paging exceptions.

Table 2.7. Conditions Preventing Instruction Restart

1. An instruction causes a task switch to a task whose Task State Segment is **partially** 'not present' (An entirely 'not present' TSS is restartable). Partially present TSS's can be avoided either by keeping the TSS's of such tasks present in memory, or by aligning TSS segments to reside entirely within a single 4K page (for TSS segments of 4K bytes or less).
2. A coprocessor operand wraps around the top of a 64K-byte segment or a 4G-byte segment, and spans three pages, and the page holding the middle portion of the operand is 'not present'. This condition can be avoided by starting **at a page boundary** any segments containing coprocessor operands if the segments are approximately 64K-200 bytes or larger (i.e. large enough for wraparound of the coprocessor operand to possibly occur).

Note that these conditions are avoided by using the operating system designs mentioned in this table.

Table 2.8. Register Values after Reset

Flag Word (EFLAGS)	uuuu0002H	Note 1
Machine Status Word (CR0)	uuuuuu10H	
Instruction Pointer (EIP)	0000FFF0H	
Code Segment (CS)	F000H	Note 2
Data Segment (DS)	0000H	Note 3
Stack Segment (SS)	0000H	
Extra Segment (ES)	0000H	Note 3
Extra Segment (FS)	0000H	
Extra Segment (GS)	0000H	
EAX register	0000H	Note 4
EDX register	component and stepping ID	Note 5
All other registers	undefined	Note 6

NOTES:

1. EFLAG Register. The upper 14 bits of the EFLAGS register are undefined, all defined flag bits are zero.
2. The Code Segment Register (CS) will have its Base Address set to 0FFFF000H and Limit set to 0FFFFH.
3. The Data and Extra Segment Registers (DS, ES) will have their Base Address set to 00000000H and Limit set to 0FFFFH.
4. If self-test is selected, the EAX register should contain a 0 value. If a value of 0 is not found then the self-test has detected a flaw in the part.
5. EDX register always holds component and stepping identifier.
6. All undefined bits are Intel Reserved and should not be used.

DOUBLE FAULT

A Double Fault (exception 8) results when the processor attempts to invoke an exception service routine for the segment exceptions (10, 11, 12 or 13), but in the process of doing so detects an exception **other than** a Page Fault (exception 14).

One other cause of generating a Double Fault is the 386 SX Microprocessor detecting any other exception when it is attempting to invoke the Page Fault (exception 14) service routine (for example, if a Page Fault is detected when the 386 SX Microprocessor attempts to invoke the Page Fault service routine). Of course, in any functional system, not only in 386 SX Microprocessor-based systems, the entire page fault service routine must remain 'present' in memory.

2.8 Reset and Initialization

When the processor is initialized or Reset the registers have the values shown in Table 2.8. The 386 SX Microprocessor will then start executing instructions near the top of physical memory, at location 0FFFF0H. When the first Intersegment Jump or Call is executed, address lines A₂₀-A₂₃ will drop LOW for CS-relative memory cycles, and the 386 SX Microprocessor will only execute instructions in the lower one megabyte of physical memory. This allows the system designer to use a shadow ROM at the top of physical memory to initialize the system and take care of Resets.

RESET forces the 386 SX Microprocessor to terminate all execution and local bus activity. No instruction execution or bus activity will occur as long as Reset is active. Between 350 and 450 CLK₂ periods after Reset becomes inactive, the 386 SX Microprocessor will start executing instructions at the top of physical memory.

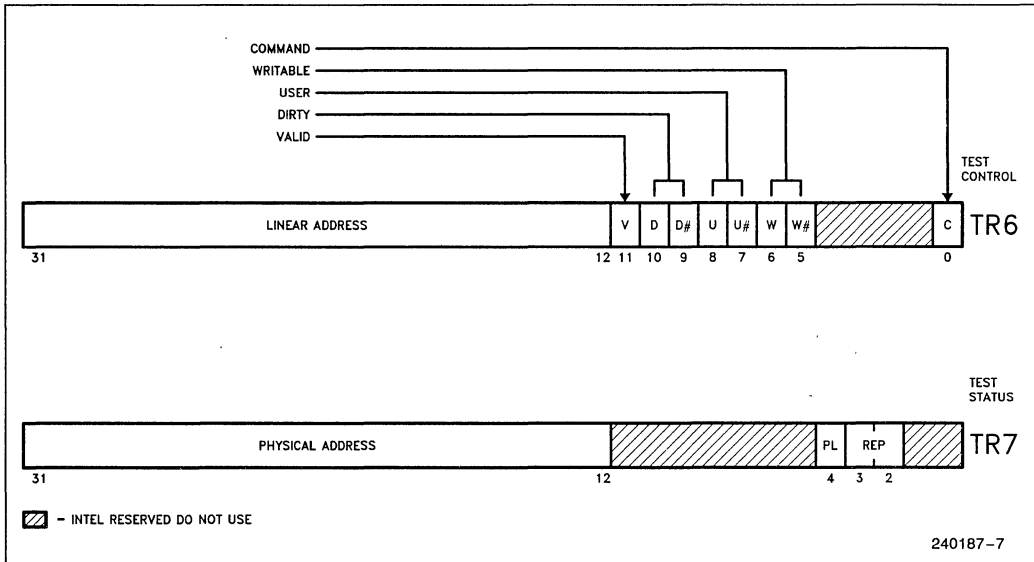
2.9 Testability

The 386 SX Microprocessor, like the 386 Microprocessor, offers testability features which include a self-test and direct access to the page translation cache.

SELF-TEST

The 386 SX Microprocessor has the capability to perform a self-test. The self-test checks the function of all of the Control ROM and most of the non-random logic of the part. Approximately one-half of the 386 SX Microprocessor can be tested during self-test.

Self-Test is initiated on the 386 SX Microprocessor when the RESET pin transitions from HIGH to LOW, and the BUSY# pin is LOW. The self-test takes about 2²⁰ clocks, or approximately 33 milliseconds with a 16 MHz 386 SX CPU. At the completion of self-test the processor performs reset and begins normal operation. The part has successfully passed self-test if the contents of the EAX are zero. If the results of the EAX are not zero then the self-test has detected a flaw in the part.


Figure 2.6. Test Registers

TLB TESTING

The 386 SX Microprocessor also provides a mechanism for testing the Translation Lookaside Buffer (TLB) if desired. This particular mechanism may not be continued in the same way in future processors.

There are two TLB testing operations: 1) writing entries into the TLB, and, 2) performing TLB lookups. Two Test Registers, shown in Figure 2.6, are provided for the purpose of testing. TR6 is the “test command register”, and TR7 is the “test data register”. For a more detailed explanation of testing the TLB, see the 386™ SX Microprocessor Programmer’s Reference Manual.

2.10 Debugging Support

The 386 SX Microprocessor provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

1. The code execution breakpoint opcode (0CCH).
2. The single-step capability provided by the TF bit in the flag register.
3. The code and data breakpoint capability provided by the Debug Registers DR0–3, DR6, and DR7.

BREAKPOINT INSTRUCTION

A single-byte software interrupt (Int 3) breakpoint instruction is available for use by software debuggers.

The breakpoint opcode is 0CCh, and generates an exception 3 trap when executed.

SINGLE-STEP TRAP

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1.

DEBUG REGISTERS

The Debug Registers are an advanced debugging feature of the 386 SX Microprocessor. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT 3 breakpoint opcode.

The 386 SX Microprocessor contains six Debug Registers, consisting of four breakpoint address registers and two breakpoint control registers. Initially after reset, breakpoints are in the disabled state; therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are auto-vectored to exception 1. Figure 2.7 shows the breakpoint status and control registers.

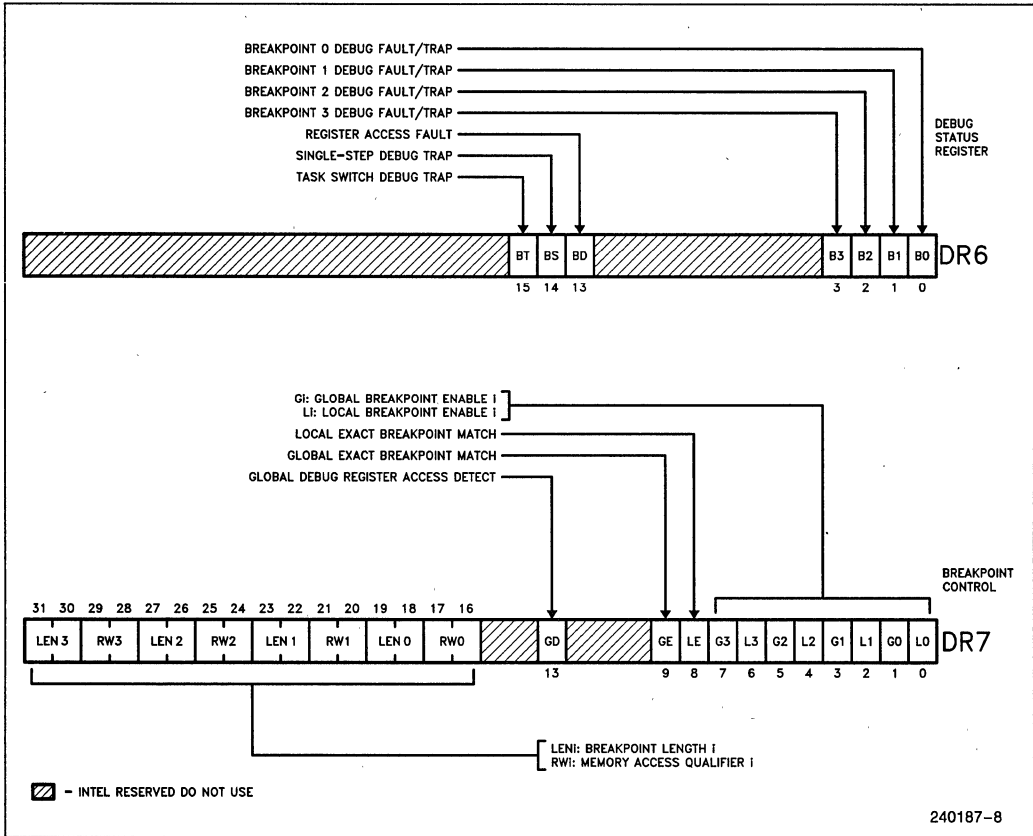


Figure 2.7. Debug Registers

3.0 REAL MODE ARCHITECTURE

When the processor is reset or powered up it is initialized in Real Mode. Real Mode has the same base architecture as the 8086, but allows access to the 32-bit register set of the 386 SX Microprocessor. The addressing mechanism, memory size, and interrupt handling are all identical to the Real Mode on the 80286.

The default operand size in Real Mode is 16 bits, as in the 8086. In order to use the 32-bit registers and addressing modes, override prefixes must be used. In addition, the segment size on the 386 SX Microprocessor in Real Mode is 64K bytes so 32-bit addresses must have a value less than 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode operation.

3.1 Memory Addressing

In Real Mode the linear addresses are the same as physical addresses (paging is not allowed). Physical addresses are formed in Real Mode by adding the contents of the appropriate segment register which is shifted left by four bits to an effective address. This addition results in a 20-bit physical address or a 1 megabyte address space. Since segment registers are shifted left by 4 bits, Real Mode segments always start on 16-byte boundaries.

All segments in Real Mode are exactly 64K bytes long, and may be read, written, or executed. The 386 SX Microprocessor will generate an exception 13 if a data operand or instruction fetch occurs past the end of a segment.

Table 3.1. Exceptions in Real Mode

Function	Interrupt Number	Related Instructions	Return Address Location
Interrupt table limit too small	8	INT vector is not within table limit	Before Instruction
CS, DS, ES, FS, GS Segment overrun exception	13	Word memory reference with offset = 0FFFFH. an attempt to execute past the end of CS segment.	Before Instruction
SS Segment overrun exception	12	Stack Reference beyond offset = 0FFFFH	Before Instruction

3.2 Reserved Locations

There are two fixed areas in memory which are reserved in Real address mode: the system initialization area and the interrupt table area. Locations 00000H through 003FFH are reserved for interrupt vectors. Each one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations 0FFFFFF0H through 0FFFFFFFH are reserved for system initialization.

3.3 Interrupts

Many of the exceptions discussed in section 2.7 are not applicable to Real Mode operation; in particular, exceptions 10, 11 and 14 do not occur in Real Mode. Other exceptions have slightly different meanings in Real Mode; Table 3.1 identifies these exceptions.

3.4 Shutdown and Halt

The HLT instruction stops program execution and prevents the processor from using the local bus until restarted. Either NMI, INTR with interrupts enabled (IF = 1), or RESET will force the 386 SX Microprocessor out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

Shutdown will occur when a severe error is detected that prevents further processing. In Real Mode, shutdown can occur under two conditions:

1. An interrupt or an exception occurs (Exceptions 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table.
2. A CALL, INT or PUSH instruction attempts to wrap around the stack segment when SP is not even.

An NMI input can bring the processor out of shutdown if the Interrupt Descriptor Table limit is large enough to contain the NMI interrupt vector (at least

000FH) and the stack has enough room to contain the vector and flag information (i.e. SP is greater than 0005H). Otherwise, shutdown can only be exited by a processor reset.

3.5 LOCK operation

The LOCK prefix on the 386 SX Microprocessor, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the 386 SX Microprocessor in Protected Mode and Virtual 8086 Mode. The LOCK prefix is not supported during repeat string instructions.

The only instruction forms where the LOCK prefix is legal on the 386 SX Microprocessor are shown in Table 3.2.

Table 3.2. Legal Instructions for the LOCK Prefix

Opcode	Operands (Dest, Source)
BIT Test and SET/RESET /COMPLEMENT	Mem, Reg/Immediate
XCHG	Reg, Mem
XCHG	Mem, Reg
ADD, OR, ADC, SBB, AND, SUB, XOR	Mem, Reg/Immediate
NOT, NEG, INC, DEC	Mem

An exception 6 will be generated if a LOCK prefix is placed before any instruction form or opcode not listed above. The LOCK prefix allows indivisible read/modify/write operations on memory operands using the instructions above.

The LOCK prefix is not IOPL-sensitive on the 386 SX Microprocessor. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed in Table 3.2.

4.0 PROTECTED MODE ARCHITECTURE

The complete capabilities of the 386 SX Microprocessor are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to four gigabytes (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 terabytes (2^{46} bytes)). In addition, Protected Mode allows the 386 SX Microprocessor to run all of the existing 386 DX CPU (using only 16 megabytes of physical memory), 80286 and 8086 CPU's software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions specially optimized for supporting multitasking operating systems. The base architecture of the 386 SX Microprocessor remains the same; the registers, instructions, and addressing modes described in the previous sections are retained. The main difference between Protected Mode and Real Mode from a programmer's viewpoint is the increased address space and a different addressing mechanism.

4.1 Addressing Mechanism

Like Real Mode, Protected Mode uses two components to form the logical address; a 16-bit selector is used to determine the linear base address of a segment, the base address is added to a 32-bit effective address to form a 32-bit linear address. The linear address is then either used as a 24-bit physical address, or if paging is enabled the paging mechanism maps the 32-bit linear address into a 24-bit physical address.

The difference between the two modes lies in calculating the base address. In Protected Mode, the selector is used to specify an index into an operating system defined table (see Figure 4.1). The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base address obtained from the table to the offset.

Paging provides an additional memory management mechanism which operates only in Protected Mode. Paging provides a means of managing the very large segments of the 386 SX Microprocessor, as paging operates beneath segmentation. The page mechanism translates the protected linear address which comes from the segmentation unit into a physical address. Figure 4.2 shows the complete 386 SX Microprocessor addressing mechanism with paging enabled.

4.2 Segmentation

Segmentation is one method of memory management. Segmentation provides the basis for protection. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about each segment is stored in an 8 byte data structure called a descriptor. All of the descriptors in a system are contained in descriptor tables which are recognized by hardware.

TERMINOLOGY

The following terms are used throughout the discussion of descriptors, privilege levels and protection:

- PL:** Privilege Level—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged.
- RPL:** Requestor Privilege Level—The privilege level of the original supplier of the selector. RPL is determined by the least two significant bits of a selector.
- DPL:** Descriptor Privilege Level—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.
- CPL:** Current Privilege Level—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.
- EPL:** Effective Privilege Level—The effective privilege level is the least privileged of the RPL and the DPL. EPL is the numerical maximum of RPL and DPL.
- Task:** One instance of the execution of a program. Tasks are also referred to as processes.

DESCRIPTOR TABLES

The descriptor tables define all of the segments which are used in a 386 SX Microprocessor system. There are three types of tables which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays and can vary in size from 8 bytes to 64K bytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address and the 16-bit limit of each table.

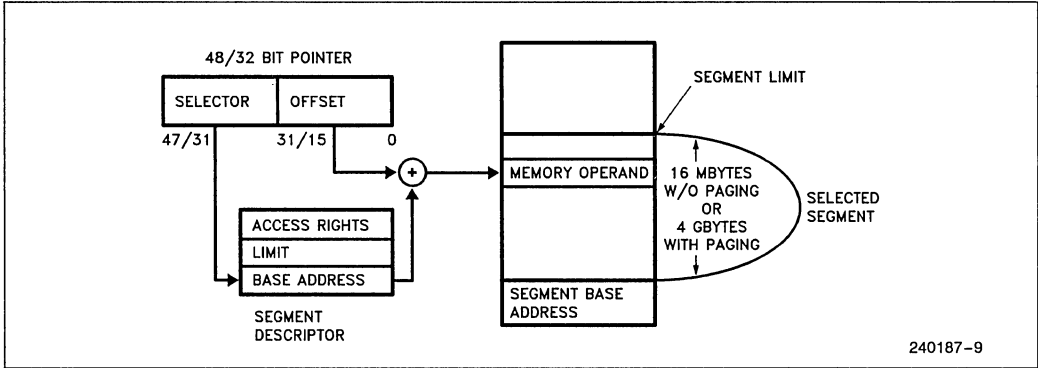


Figure 4.1. Protected Mode Addressing

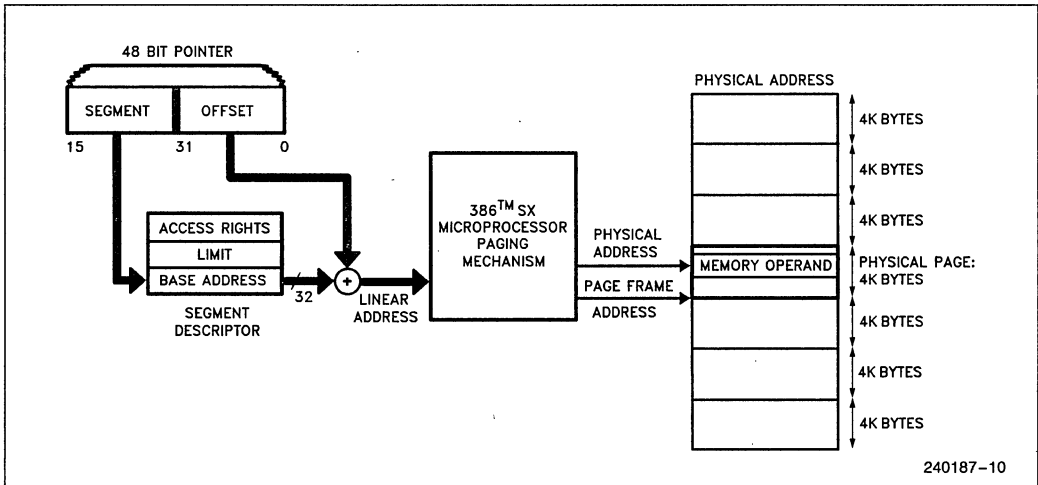


Figure 4.2. Paging and Segmentation

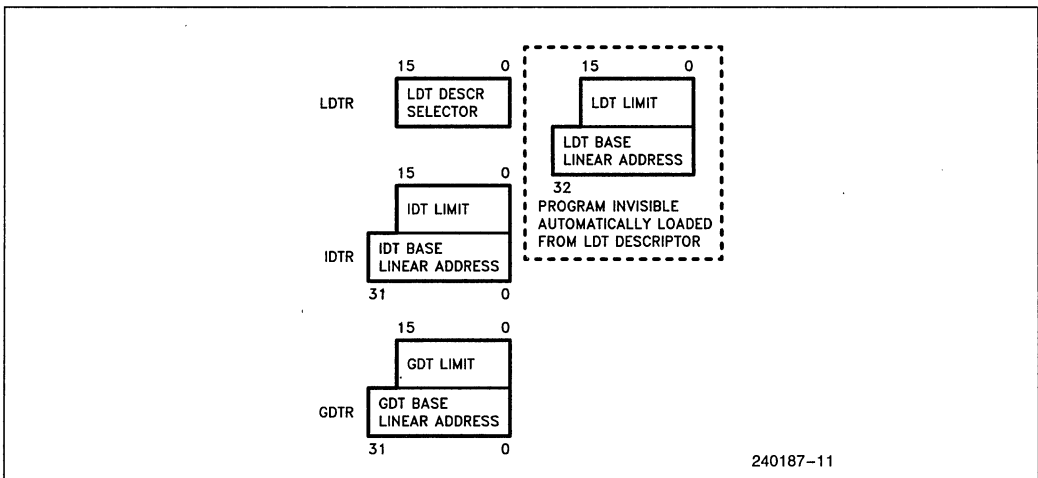


Figure 4.3. Descriptor Table Registers

Each of the tables has a register associated with it: GDTR, LDTR, and IDTR; see Figure 2.1. The LGDT, LLDT, and LIDT instructions load the base and limit of the Global, Local, and Interrupt Descriptor Tables into the appropriate register. The SGDT, SLDT, and SIDT store the base and limit values. These are privileged instructions.

Global Descriptor Table

The Global Descriptor Table (GDT) contains descriptors which are available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for interrupt and trap descriptors. Every 386 SX CPU system contains a GDT.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

Local Descriptor Table

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This provides both isolation and protection for a task's segments while still allowing global data to be shared among tasks.

Unlike the 6-byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT (see figure 2.1).

Interrupt Descriptor Table

The third table needed for 386 SX Microprocessor systems is the Interrupt Descriptor Table. The IDT contains the descriptors which point to the location of the up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced by INT instructions, external interrupt vectors, and exceptions.

DESCRIPTORS

The object to which the segment selector points to is called a descriptor. Descriptors are eight byte quantities which contain attributes about a given region of linear address space. These attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or 32-bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 4.4 shows the general format of a descriptor. All segments on the 386 SX Microprocessor have three attribute fields in common: the P bit, the DPL bit, and the S bit. The P

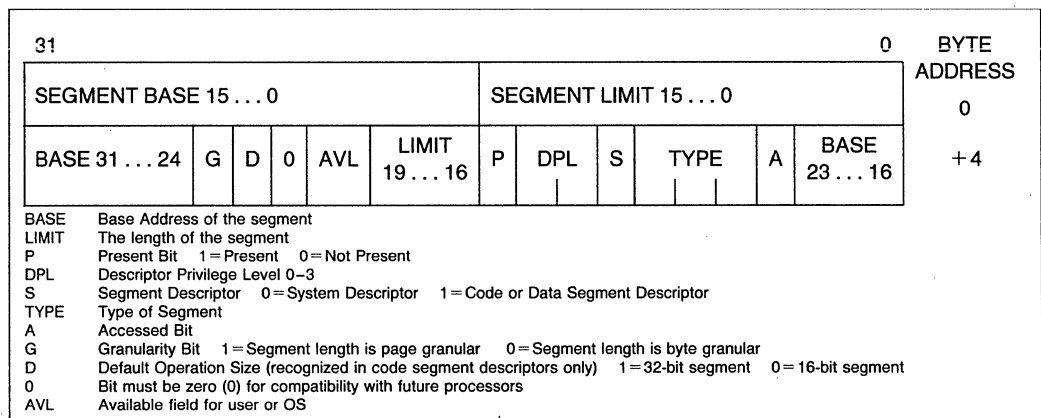


Figure 4.4. Segment Descriptors

(Present) Bit is 1 if the segment is loaded in physical memory. If P=0 then any attempt to access this segment causes a not present exception (exception 11). The Descriptor Privilege Level, DPL, is a two bit field which specifies the protection level, 0-3, associated with a segment.

The 386 SX Microprocessor has two main categories of segments: system segments and non-system segments (for code and data). The segment bit, S, determines if a given segment is a system segment

or a code or data segment. If the S bit is 1 then the segment is either a code or data segment; if it is 0 then the segment is a system segment.

Code and Data Descriptors (S = 1)

Figure 4.5 shows the general format of a code and data descriptor and Table 4.1 illustrates how the bits in the Access Right Byte are interpreted.

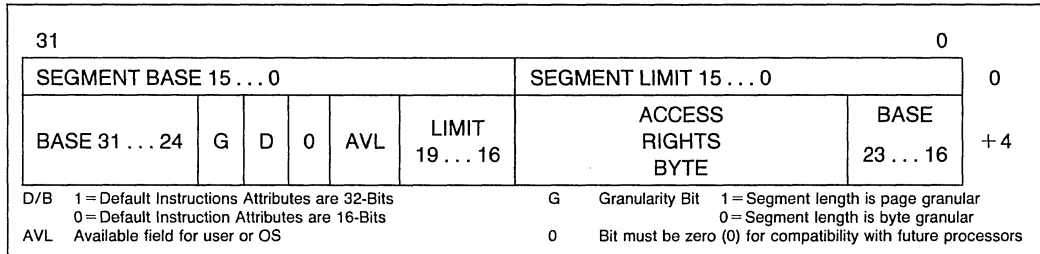


Figure 4.5. Code and Data Descriptors

Table 4.1. Access Rights Byte Definition for Code and Data Descriptors

Bit Position	Name	Function					
7	Present (P)	P = 1 Segment is mapped into physical memory. P = 0 No mapping to physical memory exists, base and limit are not used.					
6-5	Descriptor Privilege Level (DPL)	Segment privilege attribute used in privilege tests.					
4	Segment Descriptor (S)	S = 1 Code or Data (includes stacks) segment descriptor S = 0 System Segment Descriptor or Gate Descriptor					
3 2 1	Executable (E) Expansion Direction (ED) Writable (W)	<table style="border: none;"> <tr> <td style="border: none;">E = 0 Descriptor type is data segment:</td> <td rowspan="4" style="border: none; vertical-align: middle;">} If Data Segment (S = 1, E = 0)</td> </tr> <tr> <td style="border: none;">ED = 0 Expand up segment, offsets must be ≤ limit.</td> </tr> <tr> <td style="border: none;">ED = 1 Expand down segment, offsets must be > limit.</td> </tr> <tr> <td style="border: none;">W = 0 Data segment may not be written into. W = 1 Data segment may be written into.</td> </tr> </table>	E = 0 Descriptor type is data segment:	} If Data Segment (S = 1, E = 0)	ED = 0 Expand up segment, offsets must be ≤ limit.	ED = 1 Expand down segment, offsets must be > limit.	W = 0 Data segment may not be written into. W = 1 Data segment may be written into.
E = 0 Descriptor type is data segment:	} If Data Segment (S = 1, E = 0)						
ED = 0 Expand up segment, offsets must be ≤ limit.							
ED = 1 Expand down segment, offsets must be > limit.							
W = 0 Data segment may not be written into. W = 1 Data segment may be written into.							
3 2 1	Executable (E) Conforming (C) Readable (R)	<table style="border: none;"> <tr> <td style="border: none;">E = 1 Descriptor type is code segment:</td> <td rowspan="3" style="border: none; vertical-align: middle;">} If Code Segment (S = 1, E = 1)</td> </tr> <tr> <td style="border: none;">C = 1 Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.</td> </tr> <tr> <td style="border: none;">R = 0 Code segment may not be read. R = 1 Code segment may be read.</td> </tr> </table>	E = 1 Descriptor type is code segment:	} If Code Segment (S = 1, E = 1)	C = 1 Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.	R = 0 Code segment may not be read. R = 1 Code segment may be read.	
E = 1 Descriptor type is code segment:	} If Code Segment (S = 1, E = 1)						
C = 1 Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.							
R = 0 Code segment may not be read. R = 1 Code segment may be read.							
0	Accessed (A)	A = 0 Segment has not been accessed. A = 1 Segment selector has been loaded into segment register or used by selector test instructions.					

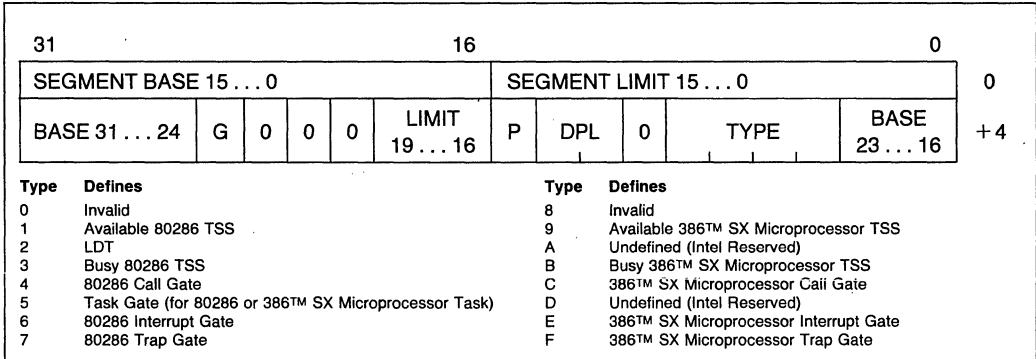


Figure 4.6. System Descriptors

Code and data segments have several descriptor fields in common. The accessed bit, A, is set whenever the processor accesses a descriptor. The granularity bit, G, specifies if a segment length is byte-granular or page-granular.

System Descriptor Formats (S=0)

System segments describe information about operating system tables, tasks, and gates. Figure 4.6 shows the general format of system segment descriptors, and the various types of system segments. 386 SX system descriptors (which are the same as 386 DX CPU system descriptors) contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.

Differences Between 386™ SX Microprocessor and 80286 Descriptors

In order to provide operating system compatibility with the 80286 the 386 SX CPU supports all of the 80286 segment descriptors. The 80286 system segment descriptors contain a 24-bit base address and 16-bit limit, while the 386 SX CPU system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for 386 SX CPU call gates.

Selector Fields

A selector in Protected Mode has three fields: Local or Global Descriptor Table indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 4.7. The TI bit selects either the Global Descriptor Table or the Local Descriptor Table. The Index selects one of 8k descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

Segment Descriptor Cache

In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Since descriptor caches only change when a segment register is changed, programs which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

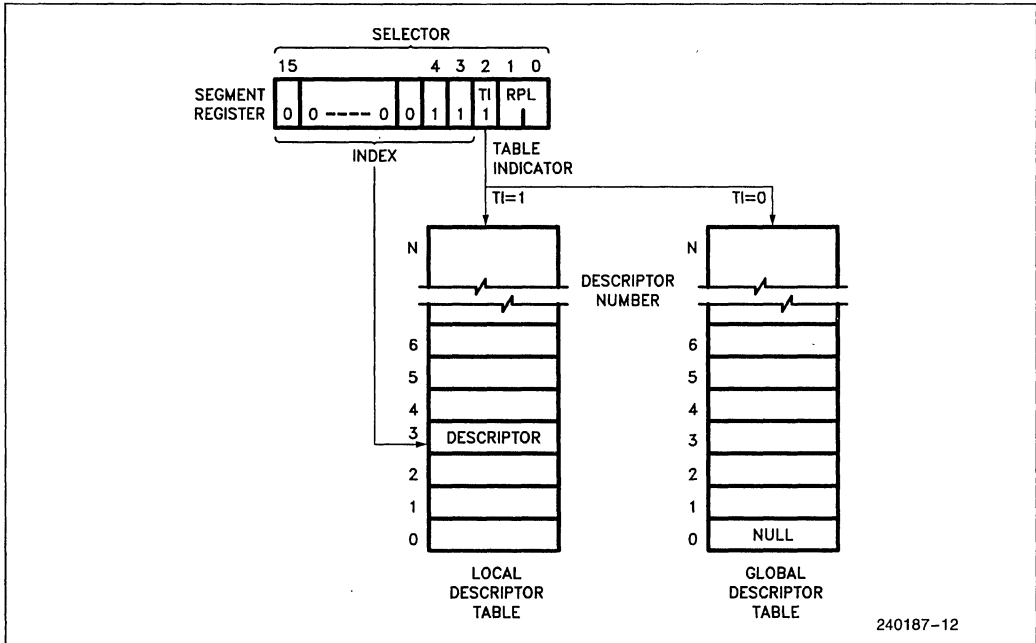


Figure 4.7. Example Descriptor Selection

240187-12

4.3 Protection

The 386 SX Microprocessor has four levels of protection which are optimized to support a multi-tasking operating system and to isolate and protect user programs from each other and the operating system. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. The 386 SX Microprocessor also offers an additional type of protection on a page basis when paging is enabled.

The four-level hierarchical privilege system is an extension of the user/supervisor privilege mode commonly used by minicomputers. The user/supervisor mode is fully supported by the 386 SX Microprocessor paging mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged level.

RULES OF PRIVILEGE

The 386 SX Microprocessor controls access to both data and procedures between levels of a task, according to the following rules.

- Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p**.
- A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p**.

PRIVILEGE LEVELS

At any point in time, a task on the 386 SX Microprocessor always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies what the task's privilege level is. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. Thus, an application program running at PL=3 may call an operating system routine at PL=1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level of the task for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (numerically larger) level of a task's CPL and a selector's RPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

Table 4.2. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL	CALL	Call Gate	GDT/LDT
	Interrupt instruction Exception External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag register) = 0

**NT (Nested Task bit of flag register) = 1

I/O Privilege

The I/O privilege level (IOPL) lets the operating system code executing at CPL = 0 define the least privileged level at which I/O instructions can be used. An exception 13 (General Protection Violation) is generated if an I/O instruction is attempted when the CPL of the task is less privileged than the IOPL. The IOPL is stored in bits 13 and 14 of the EFLAGS register. The following instructions cause an exception 13 if the CPL is greater than IOPL: IN, INS, OUT, OUTS, STI, CLI, LOCK prefix.

Descriptor Access

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL as described above.

Any time an instruction loads a data segment register (DS, ES, FS, GS) the 386 SX Microprocessor makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segment or readable code segments.

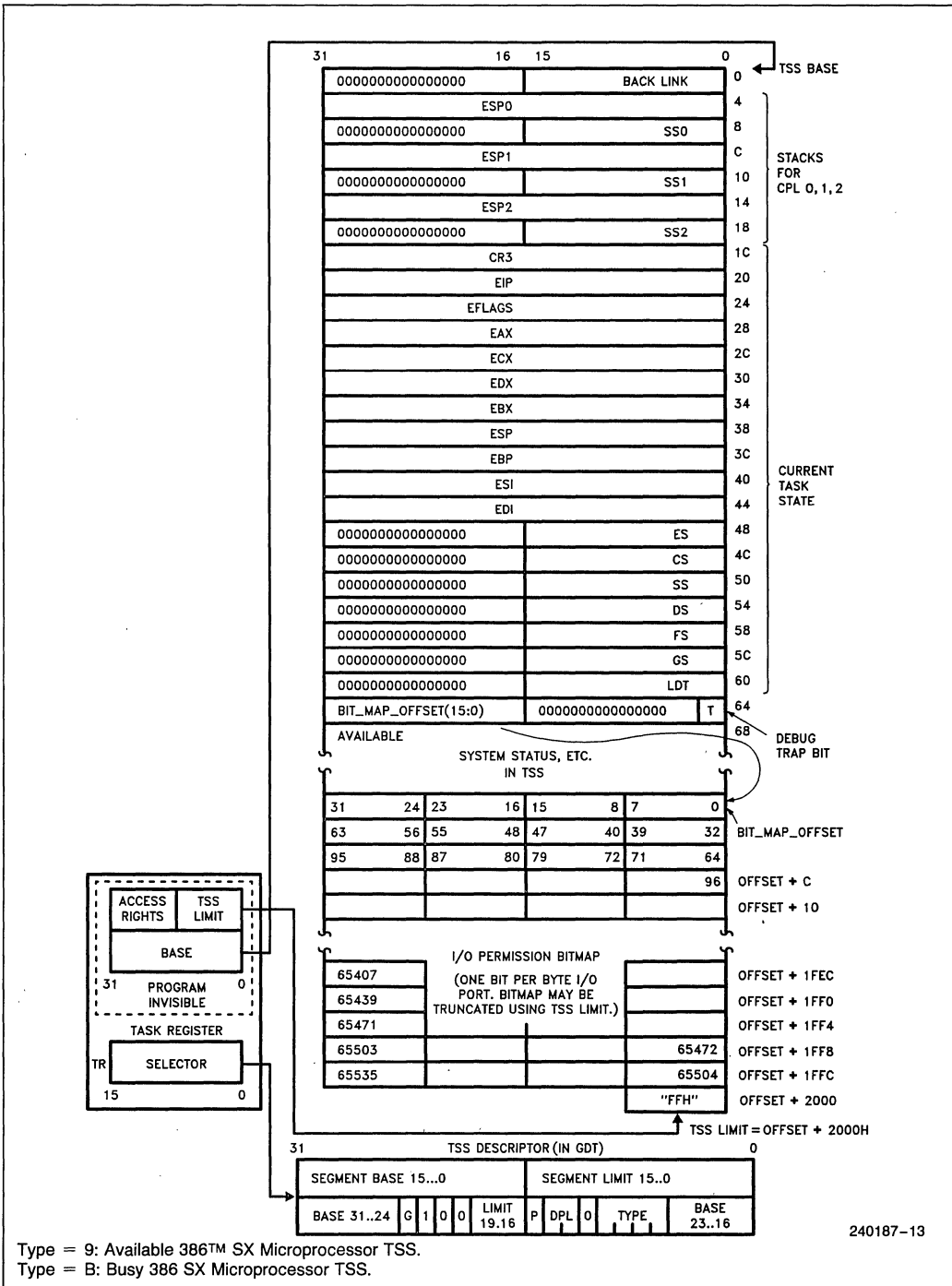
Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL, an exception 13 (general protection fault) is generated.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data segments. The DPL and RPL must equal the CPL of all other descriptor types or a privilege level violation will cause an exception 13. A stack not present fault causes an exception 12.

PRIVILEGE LEVEL TRANSFERS

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 4.2. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only by control transfers, using gates, task switches, and interrupt or trap gates.

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13.



Type = 9: Available 386™ SX Microprocessor TSS.
 Type = B: Busy 386 SX Microprocessor TSS.

240187-13

Figure 4.8. 386™ SX Microprocessor TSS and TSS Registers

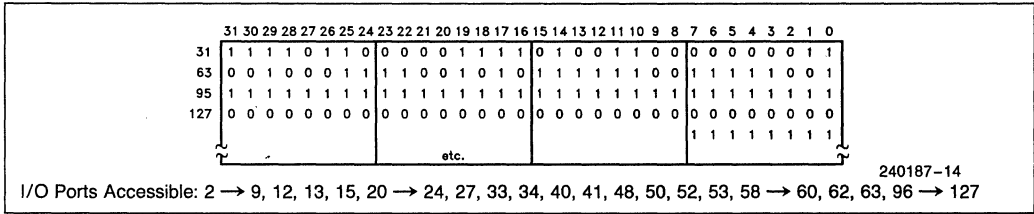


Figure 4.9. Sample I/O Permission Bit Map

CALL GATES

Gates provide protected indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures.

TASK SWITCHING

A very important attribute of any multi-tasking/multi-user operating system is its ability to rapidly switch between tasks or processes. The 386 SX Microprocessor directly supports this operation by providing a task switch instruction in hardware. The task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task. Like transfer of control by gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 4.8) containing the entire execution state. A task gate descriptor contains a TSS selector. The 386 SX Microprocessor supports both 286 and 386 SX CPU TSSs. The limit of a 386 SX Microprocessor TSS must be greater than 64H (2BH for a 286 TSS), and can be as large as 16 megabytes. In the additional TSS space, the operating system is free to store additional information such as the reason the task is inactive, time the task has spent running, or open files belonging to the task.

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 386 SX Microprocessor called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TSS descriptor are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to

the task which was interrupted. The currently executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CRO) give information about the state of a task which is useful to the operating system. The Nested Task bit, NT, controls the function of the IRET instruction. If NT=0 the IRET instruction performs the regular return. If NT=1 IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT (The NT bit will be restored after execution of the interrupt handler). NT may also be set or cleared by POPF or IRET instructions.

The 386 SX Microprocessor task state segment is marked busy by changing the descriptor type field from TYPE 9 to TYPE 0BH. A 286 TSS is marked busy by changing the descriptor type field from TYPE 1 to TYPE 3. Use of a selector that references a busy task state segment causes an exception 13.

The VM (Virtual Mode) bit is used to indicate if a task is a Virtual 8086 task. If VM=1 then the tasks will use the Real Mode addressing mechanism. The virtual 8086 environment is only entered and exited by a task switch.

The coprocessor's state is not automatically saved when a task switch occurs. The Task Switched Bit, TS, in the CRO register helps deal with the coprocessor's state in a multi-tasking environment. Whenever the 386 SX Microprocessor switches task, it sets the TS bit. The 386 SX Microprocessor detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the coprocessor.

The T bit in the 386 SX Microprocessor TSS indicates that the processor should generate a debug exception when switching to a task. If T=1 then upon entry to a new task a debug exception 1 will be generated.

INITIALIZATION AND TRANSITION TO PROTECTED MODE

Since the 386 SX Microprocessor begins executing in Real Mode immediately after RESET it is necessary to initialize the system tables and registers with the appropriate values. The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256 bytes long, and the GDT must contain descriptors for the initial code and data segments.

Protected Mode is enabled by loading CR0 with PE bit set. This can be accomplished by using the **MOV CR0, R/M** instruction. After enabling Protected Mode, the next instruction should execute an inter-segment JMP to load the CS register and flush the instruction decode queue. The final step is to load all of the data segment registers with the initial selector values.

An alternate approach to entering Protected Mode is to use the built in task-switch to load all of the registers. In this case the GDT would contain two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode would jump to the TSS causing a task switch and loading all of the registers with the values stored in the TSS. The Task State Segment Register should be initialized to point to a valid TSS descriptor.

4.4 Paging

Paging is another type of memory management useful for virtual memory multi-tasking operating systems. Unlike segmentation, which modularizes programs and data into variable length segments, paging divides programs into multiple uniform size pages. Pages bear no direct relation to the logical structure of a program. While segment selectors can be considered the logical 'name' of a program module or data structure, a page most likely corresponds to only a portion of a module or data structure.

PAGE ORGANIZATION

The 386 SX Microprocessor uses two levels of tables to translate the linear address (from the segmentation unit) into a physical address. There are three components to the paging mechanism of the 386 SX Microprocessor: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the 386 SX Microprocessor paging mechanism are the same size, namely 4K bytes. A uniform size for all of the elements simplifies memory allocation and reallocation schemes, since there is no problem with memory fragmentation. Figure 4.10 shows how the paging mechanism works.

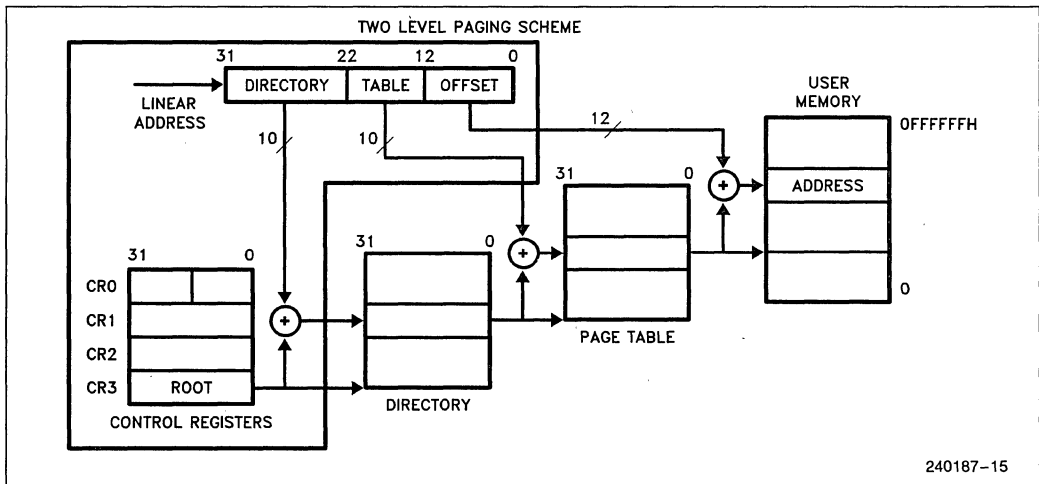


Figure 4.10. Paging Mechanism

	31		12	11	10	9	8	7	6	5	4	3	2	1	0
PAGE TABLE ADDRESS 31..12				System Software Defineable			0	0	D	A	0	0	U	R	P
													S	W	

Figure 4.11. Page Directory Entry (Points to Page Table)

31	12	11	10	9	8	7	6	5	4	3	2	1	0	
PAGE FRAME ADDRESS 31..12				System Software Defineable		0	0	D	A	0	0	U — S	R — W	P

Figure 4.12. Page Table Entry (Points to Page)

Page Fault Register

CR2 is the Page Fault Linear Address register. It holds the 32-bit linear address which caused the last Page Fault detected.

Page Descriptor Base Register

CR3 is the Page Directory Physical Base Address Register. It contains the physical starting address of the Page Directory (this value is truncated to a 24-bit value associated with the 386 SX CPU's 16 megabyte physical memory limitation). The lower 12 bits of CR3 are always zero to ensure that the Page Directory is always page aligned. Loading it with a **MOV CR3, reg** instruction causes the page table entry cache to be flushed, as will a task switch through a TSS which changes the value of CR0.

Page Directory

The Page Directory is 4k bytes long and allows up to 1024 page directory entries. Each page directory entry contains information about the page table and the address of the next level of tables, the Page Tables. The contents of a Page Directory Entry are shown in figure 4.11. The upper 10 bits of the linear address (A₃₁–A₂₂) are used as an index to select the correct Page Directory Entry.

The page table address contains the upper 20 bits of a 32-bit physical address that is used as the base address for the next set of tables, the page tables. The lower 12 bits of the page table address are zero so that the page table addresses appear on 4 kbyte boundaries. For a 386 DX CPU system the upper 20 bits will select one of 2²⁰ page tables, but for a 386 SX Microprocessor system the upper 20 bits only select one of 2¹² page tables. Again, this is because the 386 SX Microprocessor is limited to a 24-bit physical address and the upper 8 bits (A₂₄–A₃₁) are truncated when the address is output on its 24 address pins.

Page Tables

Each Page Table is 4K bytes long and allows up to 1024 Page table Entries. Each page table entry contains information about the Page Frame and its ad-

dress. The contents of a Page Table Entry are shown in figure 4.12. The middle 10 bits of the linear address (A₂₁–A₁₂) are used as an index to select the correct Page Table Entry.

The Page Frame Address contains the upper 20 bits of a 32-bit physical address that is used as the base address for the Page Frame. The lower 12 bits of the Page Frame Address are zero so that the Page Frame addresses appear on 4 kbyte boundaries. For an 386 DX CPU system the upper 20 bits will select one of 2²⁰ Page Frames, but for an 386 SX Microprocessor system the upper 20 bits only select one of 2¹² Page Frames. Again, this is because the 386 SX Microprocessor is limited to a 24-bit physical address space and the upper 8 bits (A₂₄–A₃₁) are truncated when the address is output on its 24 address pins.

Page Directory/Table Entries

The lower 12 bits of the Page Table Entries and Page Directory Entries contain statistical information about pages and page tables respectively. The P (Present) bit indicates if a Page Directory or Page Table entry can be used in address translation. If P=1, the entry can be used for address translation. If P=0, the entry cannot be used for translation. All of the other bits are available for use by the software. For example, the remaining 31 bits could be used to indicate where on disk the page is stored.

The A (Accessed) bit is set by the 386 SX CPU for both types of entries before a read or write access occurs to an address covered by the entry. The D (Dirty) bit is set to 1 before a write to an address covered by that page table entry occurs. The D bit is undefined for Page Directory Entries. When the P, A and D bits are updated by the 386 SX CPU, the processor generates a Read-Modify-Write cycle which locks the bus and prevents conflicts with other processors or peripherals. Software which modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multi-master systems.

The 3 bits marked system software definable in Figures 4.11 and Figure 4.12 are software definable. System software writers are free to use these bits for whatever purpose they wish.

PAGE LEVEL PROTECTION (R/W, U/S BITS)

The 386 SX Microprocessor provides a set of protection attributes for paging systems. The paging mechanism distinguishes between two levels of protection: User, which corresponds to level 3 of the segmentation based protection, and supervisor which encompasses all of the other protection levels (0, 1, 2). Programs executing at Level 0, 1 or 2 bypass the page protection, although segmentation-based protection is still enforced by the hardware.

The U/S and R/W bits are used to provide User/Supervisor and Read/Write protection for individual pages or for all pages covered by a Page Table Directory Entry. The U/S and R/W bits in the second level Page Table Entry apply only to the page described by that entry. While the U/S and R/W bits in the first level Page Directory Table apply to all pages described by the page table pointed to by that directory entry. The U/S and R/W bits for a given page are obtained by taking the most restrictive of the U/S and R/W from the Page Directory Table Entries and using these bits to address the page.

TRANSLATION LOOKASIDE BUFFER

The 386 SX Microprocessor paging hardware is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the processor was required to access two levels of tables for every memory reference. To solve this problem, the 386 SX Microprocessor keeps a cache of the most recently accessed pages, this cache is called the Translation Lookaside Buffer (TLB). The TLB is a four-way set associative 32-entry page table cache. It automatically keeps the most commonly used page table entries in the processor. The 32-entry TLB coupled with a 4K page size results in coverage of 128K bytes of memory addresses. For many common multi-tasking systems, the TLB will have a hit rate of greater than 98%. This means that the processor will only have to access the two-level page structure for less than 2% of all memory references.

PAGING OPERATION

The paging hardware operates in the following fashion. The paging unit hardware receives a 32-bit linear address from the segmentation unit. The upper 20 linear address bits are compared with all 32 entries in the TLB to determine if there is a match. If there is a match (i.e. a TLB hit), then the 24-bit physical address is calculated and is placed on the address bus.

If the page table entry is not in the TLB, the 386 SX Microprocessor will read the appropriate Page Directory Entry. If P = 1 on the Page Directory Entry, indicating that the page table is in memory, then the 386 SX Microprocessor will read the appropriate

Page Table Entry and set the Access bit. If P = 1 on the Page Table Entry, indicating that the page is in memory, the 386 SX Microprocessor will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. If P = 0 for either the Page Directory Entry or the Page Table Entry, then the processor will generate a page fault Exception 14.

The processor will also generate a Page Fault (Exception 14) if the memory reference violated the page protection attributes. CR2 will hold the linear address which caused the page fault. Since Exception 14 is classified as a fault, CS:EIP will point to the instruction causing the page-fault. The 16-bit error code pushed as part of the page fault handler will contain status bits which indicate the cause of the page fault.

The 16-bit error code is used by the operating system to determine how to handle the Page Fault. Figure 4.13 shows the format of the Page Fault error code and the interpretation of the bits. Even though the bits in the error code (U/S, W/R, and P) have similar names as the bits in the Page Directory/Table Entries, the interpretation of the error code bits is different. Figure 4.14 indicates what type of access caused the page fault.

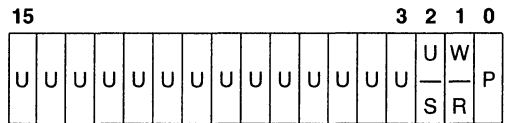


Figure 4.13. Page Fault Error Code Format

U/S: The U/S bit indicates whether the access causing the fault occurred when the processor was executing in User Mode (U/S = 1) or in Supervisor mode (U/S = 0)

W/R: The W/R bit indicates whether the access causing the fault was a Read (W/R = 0) or a Write (W/R = 1)

P: The P bit indicates whether a page fault was caused by a not-present page (P = 0), or by a page level protection violation (P = 1)

U = Undefined

U/S	W/R	Access Type
0	0	Supervisor* Read
0	1	Supervisor Write
1	0	User Read
1	1	User Write

*Descriptor table access will fault with U/S = 0, even if the program is executing at level 3.

Figure 4.14. Type of Access Causing Page Fault

OPERATING SYSTEM RESPONSIBILITIES

When the operating system enters or exits paging mode (by setting or resetting bit 31 in the CR0 register) a short JMP must be executed to flush the 386 SX Microprocessor's prefetch queue. This ensures that all instructions executed after the address mode change will generate correct addresses.

The 386 SX Microprocessor takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for setting up the initial page tables and handling any page faults. The operating system also is required to invalidate (i.e. flush) the TLB when any changes are made to any of the page table entries. The operating system must reload CR3 to cause the TLB to be flushed.

Setting up the tables is simply a matter of loading CR3 with the address of the Page Directory, and allocating space for the Page Directory and the Page Tables. The primary responsibility of the operating system is to implement a swapping policy and handle all of the page faults.

A final concern of the operating system is to ensure that the TLB cache matches the information in the paging tables. In particular, any time the operating systems sets the P (Present) bit of page table entry to zero. The TLB must be flushed by reloading CR3. Operating systems may want to take advantage of the fact that CR3 is stored as part of a TSS, to give every task or group of tasks its own set of page tables.

4.5 Virtual 8086 Environment

The 386 SX Microprocessor allows the execution of 8086 application programs in both Real Mode and in the Virtual 8086 Mode. The Virtual 8086 Mode allows the execution of 8086 applications, while still allowing the system designer to take full advantage of the 386 SX CPU's protection mechanism.

VIRTUAL 8086 ADDRESSING MECHANISM

One of the major differences between 386 SX CPU Real and Protected modes is how the segment selectors are interpreted. When the processor is executing in Virtual 8086 Mode, the segment registers are used in a fashion identical to Real Mode. The contents of the segment register are shifted left 4 bits and added to the offset to form the segment base linear address.

The 386 SX Microprocessor allows the operating system to specify which programs use the 8086 ad-

dress mechanism and which programs use Protected Mode addressing on a per task basis. Through the use of paging, the one megabyte address space of the Virtual Mode task can be mapped to anywhere in the 4 gigabyte linear address space of the 386 SX Microprocessor. Like Real Mode, Virtual Mode addresses that exceed one megabyte will cause an exception 13. However, these restrictions should not prove to be important, because most tasks running in Virtual 8086 Mode will simply be existing 8086 application programs.

PAGING IN VIRTUAL MODE

The paging hardware allows the concurrent running of multiple Virtual Mode tasks, and provides protection and operating system isolation. Although it is not strictly necessary to have the paging hardware enabled to run Virtual Mode tasks, it is needed in order to run multiple Virtual Mode tasks or to relocate the address space of a Virtual Mode task to physical address space greater than one megabyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into as many as 256 pages. Each one of the pages can be located anywhere within the maximum 16 megabyte physical address space of the 386 SX Microprocessor. In addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations. Finally, the paging hardware allows the sharing of the 8086 operating system code between multiple 8086 applications.

PROTECTION AND I/O PERMISSION BIT MAP

All Virtual Mode programs execute at privilege level 3. As such, Virtual Mode programs are subject to all of the protection checks defined in Protected Mode. This is different than Real Mode, which implicitly is executing at privilege level 0. Thus, an attempt to execute a privileged instruction in Virtual Mode will cause an exception 13 fault.

The following are privileged instructions, which may be executed only at Privilege Level 0. Attempting to execute these instructions in Virtual 8086 Mode (or anytime $CPL \geq 0$) causes an exception 13 fault:

LIDT;	MOV DRn,REG;	MOV reg,DRn;
LGDT;	MOV TRn,reg;	MOV reg,TRn;
LMSW;	MOV CRn,reg;	MOV reg,CRn;

CLTS;
HLT;

Several instructions, particularly those applying to the multitasking and the protection model, are available only in Protected Mode. Therefore, attempting to execute the following instructions in Real Mode or in Virtual 8086 Mode generates an exception 6 fault:

```
LTR;   STR;
LLDT;  SLDT;
LAR;   VERR;
LSL;   VERW;
ARPL;
```

The instructions which are IOPL sensitive in Protected Mode are:

```
IN;    STI;
OUT;   CLI;
INS;
OUTS;
REP INS;
REP OUTS;
```

In Virtual 8086 Mode the following instructions are IOPL-sensitive:

```
INT n; STI;
PUSHF; CLI;
POPF;  IRET;
```

The PUSHF, POPF, and IRET instructions are IOPL-sensitive in Virtual 8086 Mode only. This provision allows the IF flag to be virtualized to the virtual 8086 Mode program. The INT n software interrupt instruction is also IOPL-sensitive in Virtual 8086 mode. Note that the INT 3, INTO, and BOUND instructions are not IOPL-sensitive in Virtual 8086 Mode.

The I/O instructions that directly refer to addresses in the processor's I/O space are IN, INS, OUT, and OUTS. The 386 SX Microprocessor has the ability to selectively trap references to specific I/O addresses. The structure that enables selective trapping is the *I/O Permission Bit Map* in the TSS segment (see Figures 4.8 and 4.9). The I/O permission map is a bit vector. The size of the map and its location in the TSS segment are variable. The processor locates the I/O permission map by means of the **I/O map base** field in the fixed portion of the TSS. The **I/O map base** field is 16 bits wide and contains the offset of the beginning of the I/O permission map.

In protected mode when an I/O instruction (IN, INS, OUT or OUTS) is encountered, the processor first checks whether $CPL \leq IOPL$. If this condition is true, the I/O operation may proceed. If not true, the processor checks the I/O permission map (in Virtual 8086 Mode, the processor consults the map without regard for the IOPL).

Each bit in the map corresponds to an I/O port byte address; for example, the bit for port 41 is found at **I/O map base** + 5, bit offset 1. The processor tests all the bits that correspond to the I/O addresses spanned by an I/O operation; for example, a double word operation tests four bits corresponding to four adjacent byte addresses. If any tested bit is set, the processor signals a general protection exception. If all the tested bits are zero, the I/O operations may proceed.

It is not necessary for the I/O permission map to represent all the I/O addresses. I/O addresses not spanned by the map are treated as if they had one-bits in the map. The **I/O map base** should be at least one byte less than the TSS limit, the last byte beyond the I/O mapping information must contain all 1's.

Because the I/O permission map is in the TSS segment, different tasks can have different maps. Thus, the operating system can allocate ports to a task by changing the I/O permission map in the task's TSS.

IMPORTANT IMPLEMENTATION NOTE: Beyond the last byte of I/O mapping information in the I/O permission bit map **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the 386 SX CPU TSS segment (see Figure 4.8).

Interrupt Handling

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled in a unique fashion. When running in Virtual Mode all interrupts and exceptions involve a privilege change back to the host 386 SX Microprocessor operating system. The 386 SX Microprocessor operating system determines if the interrupt comes from a Protected Mode application or from a Virtual Mode program by examining the VM bit in the EFLAGS image stored on the stack.

When a Virtual Mode program is interrupted and execution passes to the interrupt routine at level 0, the VM bit is cleared. However, the VM bit is still set in the EFLAG image on the stack.

The 386 SX Microprocessor operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The 386 SX Microprocessor operating system may choose to let the 8086 operating system handle the interrupt or it may emulate the function of the interrupt handler. For example, many 8086 operating system calls are accessed by PUSHing parameters on the stack, and then executing an INT n instruction. If the IOPL is set to 0 then all INT n instructions will be intercepted by the 386 SX Microprocessor operating system.

An 386 SX Microprocessor operating system can provide a Virtual 8086 Environment which is totally transparent to the application software by intercepting and then emulating 8086 operating system's calls, and intercepting IN and OUT instructions.

Entering and Leaving Virtual 8086 Mode

Virtual 8086 mode is entered by executing a 32-bit IRET instruction at CPL=0 where the stack has a 1 in the VM bit of its EFLAGS image, or a Task Switch (at any CPL) to a 386 SX Microprocessor task whose 386 SX CPU TSS has a EFLAGS image containing a 1 in the VM bit position while the processor is executing in the Protected Mode. POPF does not affect the VM bit but a PUSHF always pushes a 0 in the VM bit.

The transition out of Virtual 8086 mode to protected mode occurs only on receipt of an interrupt or exception. In Virtual 8086 mode, all interrupts and exceptions vector through the protected mode IDT, and enter an interrupt handler in protected mode. As part of the interrupt processing the VM bit is cleared.

Because the matching IRET must occur from level 0, Interrupt or Trap Gates used to field an interrupt or exception out of Virtual 8086 mode must perform an inter-level interrupt only to level 0. Interrupt or Trap Gates through conforming segments, or through segments with DPL>0, will raise a GP fault with the CS selector as the error code.

Task Switches To/From Virtual 8086 Mode

Tasks which can execute in Virtual 8086 mode must be described by a TSS with the 386 SX CPU format (type 9 or 11 descriptor). A task switch out of virtual 8086 mode will operate exactly the same as any other task switch out of a task with a 386 SX CPU TSS. All of the programmer visible state, including the EFLAGS register with the VM bit set to 1, is stored in the TSS. The segment registers in the TSS will contain 8086 segment base values rather than selectors.

A task switch into a task described by a 386 SX CPU TSS will have an additional check to determine if the incoming task should be resumed in Virtual 8086 mode. Tasks described by 286 format TSSs cannot be resumed in Virtual 8086 mode, so no check is required there (the EFLAGS image in 286 format TSS has only the low order 16 EFLAGS bits). Before loading the segment register images from a 386 SX CPU TSS, the EFLAGS image is loaded, so that the segment registers are loaded from the TSS image as 8086 segment base values. The task is now ready to resume in Virtual 8086 mode.

Transitions Through Trap and Interrupt Gates, and IRET

A task switch is one way to enter or exit Virtual 8086 mode. The other method is to exit through a Trap or Interrupt gate, as part of handling an interrupt, and to enter as part of executing an IRET instruction. The transition out must use a 386 SX CPU Trap Gate (Type 14), or 386 SX CPU Interrupt Gate (Type 15), which must point to a non-conforming level 0 segment (DPL=0) in order to permit the trap handler to IRET back to the Virtual 8086 program. The Gate must point to a non-conforming level 0 segment to perform a level switch to level 0 so that the matching IRET can change the VM bit. 386 SX CPU gates must be used since 286 gates save only the low 16 bits of the EFLAGS register (the VM bit will not be saved). Also, the 16-bit IRET used to terminate the 286 interrupt handler will pop only the lower 16 bits from EFLAGS, and will not affect the VM bit. The action taken for a 386 SX CPU Trap or Interrupt gate if an interrupt occurs while the task is executing in virtual 8086 mode is given by the following sequence:

1. Save the EFLAGS register in a temp to push later. Turn off the VM, TF, and IF bits.
2. Interrupt and Trap gates must perform a level switch from 3 (where the Virtual 8086 Mode program executes) to level 0 (so IRET can return).
3. Push the 8086 segment register values onto the new stack, in this order: GS, FS, DS, ES. These are pushed as 32-bit quantities. Then load these 4 registers with null selectors (0).
4. Push the old 8086 stack pointer onto the new stack by pushing the SS register (as 32-bits), then pushing the 32-bit ESP register saved above.
5. Push the 32-bit EFLAGS register saved in step 1.
6. Push the old 8086 instruction onto the new stack by pushing the CS register (as 32-bits), then pushing the 32-bit EIP register.
7. Load up the new CS:EIP value from the interrupt gate, and begin execution of the interrupt routine in protected mode.

The transition out of V86 mode performs a level change and stack switch, in addition to changing back to protected mode. Also all of the 8086 segment register images are stored on the stack (behind the SS:ESP image), and then loaded with null (0) selectors before entering the interrupt handler. This will permit the handler to safely save and restore the DS, ES, FS, and GS registers as 286 selectors. This is needed so that interrupt handlers which don't care about the mode of the interrupted program can use the same prologue and epilogue code for state saving regardless of whether or not a 'native' mode or Virtual 8086 Mode program was interrupted. Restoring null selectors to these registers

before executing the IRET will cause a trap in the interrupt handler. Interrupt routines which expect or return values in the segment registers will have to obtain/return values from the 8086 register images pushed onto the new stack. They will need to know the mode of the interrupted program in order to know where to find/return segment registers, and also to know how to interpret segment register values.

The IRET instruction will perform the inverse of the above sequence. Only the extended IRET instruction (operand size=32) can be used and must be executed at level 0 to change the VM bit to 1.

1. If the NT bit in the FLAGS register is on, an intertask return is performed. The current state is stored in the current TSS, and the link field in the current TSS is used to locate the TSS for the interrupted task which is to be resumed. Otherwise, continue with the following sequence:
2. Read the FLAGS image from SS:8[ESP] into the FLAGS register. This will set VM to the value active in the interrupted routine.
3. Pop off the instruction pointer CS:EIP. EIP is popped first, then a 32-bit word is popped which contains the CS value in the lower 16 bits. If VM=0, this CS load is done as a protected mode segment load. If VM=1, this will be done as an 8086 segment load.
4. Increment the ESP register by 4 to bypass the FLAGS image which was 'popped' in step 1.
5. If VM=1, load segment registers ES, DS, FS, and GS from memory locations SS:[ESP+8], SS:[ESP+12], SS:[ESP+16], and SS:[ESP+20], respectively, where the new value of ESP stored in step 4 is used. Since VM=1, these are done as 8086 segment register loads.
Else if VM=0, check that the selectors in ES, DS, FS, and GS are valid in the interrupted routine. Null out invalid selectors to trap if an attempt is made to access through them.
6. If RPL(CS)>CPL, pop the stack pointer SS:ESP from the stack. The ESP register is popped first, followed by 32-bits containing SS in the lower 16 bits. If VM=0, SS is loaded as a protected mode segment register load. If VM=1, an 8086 segment register load is used.
7. Resume execution of the interrupted routine. The VM bit in the FLAGS register (restored from the interrupt routine's stack image in step 1) determines whether the processor resumes the interrupted routine in Protected mode or Virtual 8086 Mode.

5.0 FUNCTIONAL DATA

The 386 SX Microprocessor features a straightforward functional interface to the external hardware. The 386 SX Microprocessor has separate parallel buses for data and address. The data bus is 16-bits in width, and bi-directional. The address bus outputs 24-bit address values using 23 address lines and two byte enable signals.

The 386 SX Microprocessor has two selectable address bus cycles: address pipelined and non-address pipelined. The address pipelining option allows as much time as possible for data access by starting the pending bus cycle before the present bus cycle is finished. A non-pipelined bus cycle gives the highest bus performance by executing every bus cycle in two processor CLK cycles. For maximum design flexibility, the address pipelining option is selectable on a cycle-by-cycle basis.

The processor's bus cycle is the basic mechanism for information transfer, either from system to processor, or from processor to system. 386 SX Microprocessor bus cycles perform data transfer in a minimum of only two clock periods. The maximum transfer bandwidth at 16 MHz is therefore 16 Mbytes/sec. However, any bus cycle will be extended for more than two clock periods if external hardware withholds acknowledgement of the cycle.

The 386 SX Microprocessor can relinquish control of its local buses to allow mastership by other devices, such as direct memory access (DMA) channels. When relinquished, HLDA is the only output pin driven by the 386 SX Microprocessor, providing near-complete isolation of the processor from its system (all other output pins are in a float condition).

5.1 Signal Description Overview

Ahead is a brief description of the 386 SX Microprocessor input and output signals arranged by functional groups. Note the # symbol at the end of a signal name indicates the active, or asserted, state occurs when the signal is at a LOW voltage. When no # is present after the signal name, the signal is asserted when at the HIGH voltage level.

Example signal: M/IO# — HIGH voltage indicates Memory selected
— LOW voltage indicates I/O selected

The signal descriptions sometimes refer to AC timing parameters, such as 't₂₅ Reset Setup Time' and 't₂₆ Reset Hold Time.' The values of these parameters can be found in Table 7.4.

CLOCK (CLK2)

CLK2 provides the fundamental timing for the 386 SX Microprocessor. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock is comprised of two phases, 'phase one' and 'phase two'. Each CLK2 period is a phase of the internal clock. Figure 5.2 illustrates the relationship. If desired, the phase of the internal processor clock can be synchronized to a known phase by ensuring the falling edge of the RESET signal meets the applicable setup and hold times t_{25} and t_{26} .

DATA BUS (D₁₅-D₀)

These three-state bidirectional signals provide the general purpose data path between the 386 SX Microprocessor and other devices. The data bus outputs are active HIGH and will float during bus hold acknowledge. Data bus reads require that read-data setup and hold times t_{21} and t_{22} be met relative to CLK2 for correct operation.

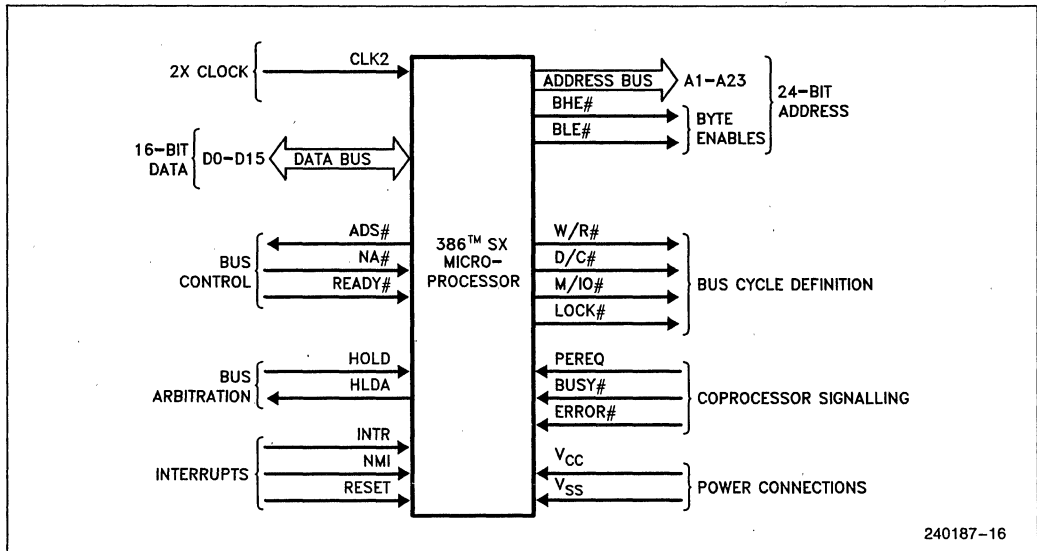


Figure 5.1. Functional Signal Groups

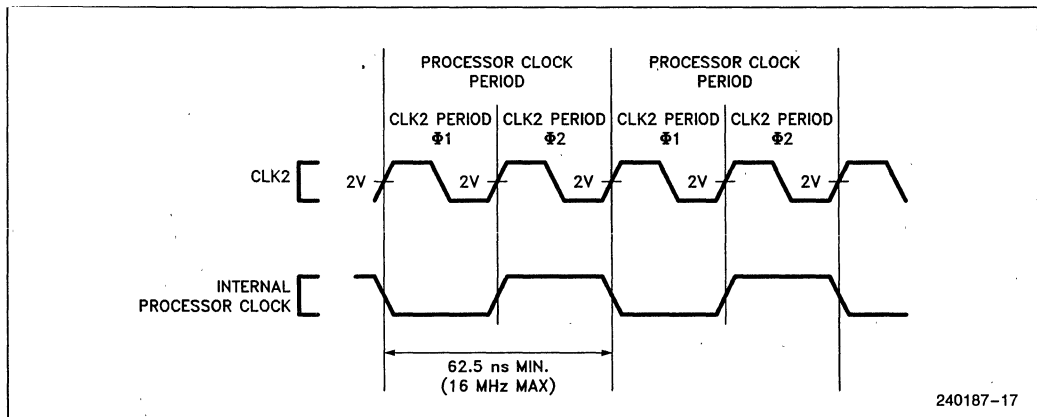


Figure 5.2. CLK2 Signal and Internal Processor Clock

ADDRESS BUS (A₂₃-A₁, BHE#, BLE#)

These three-state outputs provide physical memory addresses or I/O port addresses. A₂₃-A₁₆ are LOW during I/O transfers except for I/O transfers automatically generated by coprocessor instructions. During coprocessor I/O transfers, A₂₂-A₁₆ are driven LOW, and A₂₃ is driven HIGH so that this address line can be used by external logic to generate the coprocessor select signal. Thus, the I/O address driven by the 386 SX Microprocessor for coprocessor commands is 8000F8H, the I/O addresses driven by the 386 SX Microprocessor for coprocessor data are 8000FCH or 8000FEH for cycles to the 387™ SX.

The address bus is capable of addressing 16 megabytes of physical memory space (000000H through FFFFFFFH), and 64 kilobytes of I/O address space (000000H through 00FFFFFFH) for programmed I/O. The address bus is active HIGH and will float during bus hold acknowledge.

The Byte Enable outputs, BHE# and BLE#, directly indicate which bytes of the 16-bit data bus are involved with the current transfer. BHE# applies to D₁₅-D₈ and BLE# applies to D₇-D₀. If both BHE# and BLE# are asserted, then 16 bits of data are being transferred. See Table 5.1 for a complete decoding of these signals. The byte enables are active LOW and will float during bus hold acknowledge.

BUS CYCLE DEFINITION SIGNALS (W/R#, D/C#, M/IO#, LOCK#)

These three-state outputs define the type of bus cycle being performed: W/R# distinguishes between

write and read cycles, D/C# distinguishes between data and control cycles, M/IO# distinguishes between memory and I/O cycles, and LOCK# distinguishes between locked and unlocked bus cycles. All of these signals are active LOW and will float during bus acknowledge.

The primary bus cycle definition signals are W/R#, D/C# and M/IO#, since these are the signals driven valid as ADS# (Address Status output) becomes active. The LOCK# is driven valid at the same time the bus cycle begins, which due to address pipelining, could be after ADS# becomes active. Exact bus cycle definitions, as a function of W/R#, D/C#, and M/IO# are given in Table 5.2.

LOCK# indicates that other system bus masters are not to gain control of the system bus while it is active. LOCK# is activated on the CLK2 edge that begins the first locked bus cycle (i.e., it is not active at the same time as the other bus cycle definition pins) and is deactivated when ready is returned at the end of the last bus cycle which is to be locked. The beginning of a bus cycle is determined when READY# is returned in a previous bus cycle and another is pending (ADS# is active) or the clock in which ADS# is driven active if the bus was idle. This means that it follows more closely with the write data rules when it is valid, but may cause the bus to be locked longer than desired. The LOCK# signal may be explicitly activated by the LOCK prefix on certain instructions. LOCK# is always asserted when executing the XCHG instruction, during descriptor updates, and during the interrupt acknowledge sequence.

Table 5.1. Byte Enable Definitions

BHE#	BLE#	Function
0	0	Word Transfer
0	1	Byte transfer on upper byte of the data bus, D ₁₅ -D ₈
1	0	Byte transfer on lower byte of the data bus, D ₇ -D ₀
1	1	Never occurs

Table 5.2. Bus Cycle Definition

M/IO#	D/C#	W/R#	Bus Cycle Type	Locked?
0	0	0	Interrupt Acknowledge	Yes
0	0	1	does not occur	—
0	1	0	I/O Data Read	No
0	1	1	I/O Data Write	No
1	0	0	Memory Code Read	No
1	0	1	Halt: Shutdown: Address = 2 Address = 0 BHE# = 1 BHE# = 1 BLE# = 0 BLE# = 0	No
1	1	0	Memory Data Read	Some Cycles
1	1	1	Memory Data Write	Some Cycles

BUS CONTROL SIGNALS (ADS#, READY#, NA#)

The following signals allow the processor to indicate when a bus cycle has begun, and allow other system hardware to control address pipelining and bus cycle termination.

Address Status (ADS#)

This three-state output indicates that a valid bus cycle definition and address (W/R#, D/C#, M/IO#, BHE#, BLE# and A₂₃-A₁) are being driven at the 386 SX Microprocessor pins. ADS# is an active LOW output. Once ADS# is driven active, valid address, byte enables, and definition signals will not change. In addition, ADS# will remain active until its associated bus cycle begins (when READY# is returned for the previous bus cycle when running pipelined bus cycles). When address pipelining is utilized, maximum throughput is achieved by initiating bus cycles when ADS# and READY# are active in the same clock cycle. ADS# will float during bus hold acknowledge. See sections **Non-Pipelined Address** and **Pipelined Address** for additional information on how ADS# is asserted for different bus states.

Transfer Acknowledge (READY#)

This input indicates the current bus cycle is complete, and the active bytes indicated by BHE# and BLE# are accepted or provided. When READY# is sampled active during a read cycle or interrupt acknowledge cycle, the 386 SX Microprocessor latches the input data and terminates the cycle. When READY# is sampled active during a write cycle, the processor terminates the bus cycle.

READY# is ignored on the first bus state of all bus cycles, and sampled each bus state thereafter until asserted. READY# must eventually be asserted to acknowledge every bus cycle, including Halt Indication and Shutdown Indication bus cycles. When being sampled, READY# must always meet setup and hold times t₁₉ and t₂₀ for correct operation.

Next Address Request (NA#)

This is used to request address pipelining. This input indicates the system is prepared to accept new values of BHE#, BLE#, A₂₃-A₁, W/R#, D/C# and M/IO# from the 386 SX Microprocessor even if the end of the current cycle is not being acknowledged on READY#. If this input is active when sampled, the next address is driven onto the bus, provided the next bus request is already pending internally. NA# is ignored in CLK cycles in which ADS# or READY#

is activated. This signal is active LOW and must satisfy setup and hold times t₁₅ and t₁₆ for correct operation. See **Pipelined Address** and **Read and Write Cycles** for additional information.

BUS ARBITRATION SIGNALS (HOLD, HLDA)

This section describes the mechanism by which the processor relinquishes control of its local buses when requested by another bus master device. See **Entering and Exiting Hold Acknowledge** for additional information.

Bus Hold Request (HOLD)

This input indicates some device other than the 386 SX Microprocessor requires bus mastership. When control is granted, the 386 SX Microprocessor floats A₂₃-A₁, BHE#, BLE#, D₁₅-D₀, LOCK#, M/IO#, D/C#, W/R# and ADS#, and then activates HLDA, thus entering the bus hold acknowledge state. The local bus will remain granted to the requesting master until HOLD becomes inactive. When HOLD becomes inactive, the 386 SX Microprocessor will deactivate HLDA and drive the local bus (at the same time), thus terminating the hold acknowledge condition.

HOLD must remain asserted as long as any other device is a local bus master. External pull-up resistors may be required when in the hold acknowledge state since none of the 386 SX Microprocessor floated outputs have internal pull-up resistors. See **Resistor Recommendations** for additional information. HOLD is not recognized while RESET is active. If RESET is asserted while HOLD is asserted, RESET has priority and places the bus into an idle state, rather than the hold acknowledge (high-impedance) state.

HOLD is a level-sensitive, active HIGH, synchronous input. HOLD signals must always meet setup and hold times t₂₃ and t₂₄ for correct operation.

Bus Hold Acknowledge (HLDA)

When active (HIGH), this output indicates the 386 SX Microprocessor has relinquished control of its local bus in response to an asserted HOLD signal, and is in the bus Hold Acknowledge state.

The Bus Hold Acknowledge state offers near-complete signal isolation. In the Hold Acknowledge state, HLDA is the only signal being driven by the 386 SX Microprocessor. The other output signals or bidirectional signals (D₁₅-D₀, BHE#, BLE#, A₂₃-A₁, W/R#, D/C#, M/IO#, LOCK# and ADS#) are in a high-impedance state so the requesting bus

master may control them. These pins remain OFF throughout the time that HLDA remains active (see Table 5.3)). Pull-up resistors may be desired on several signals to avoid spurious activity when no bus master is driving them. See **Resistor Recommendations** for additional information.

When the HOLD signal is made inactive, the 386 SX Microprocessor will deactivate HLDA and drive the bus. One rising edge on the NMI input is remembered for processing after the HOLD input is negated.

Table 5.3. Output pin State During HOLD

Pin Value	Pin Names
1	HLDA
Float	LOCK#, M/IO#, D/C#, W/R#, ADS#, A ₂₃ -A ₁ , BHE#, BLE#, D ₁₅ -D ₀

In addition to the normal usage of Hold Acknowledge with DMA controllers or master peripherals, the near-complete isolation has particular attractiveness during system test when test equipment drives the system, and in hardware fault-tolerant applications.

HOLD Latencies

The maximum possible HOLD latency depends on the software being executed. The actual HOLD latency at any time depends on the current bus activity, the state of the LOCK# signal (internal to the CPU) activated by the LOCK# prefix, and interrupts. The 386 SX Microprocessor will not honor a HOLD request until the current bus operation is complete. Table 5.4 shows the types of bus operations that can affect HOLD latency, and indicates the types of delays that these operations may introduce. When considering maximum HOLD latencies, designers must select which of these bus operations are possible, and then select the maximum latency from among them.

The 386 SX Microprocessor breaks 32-bit data or I/O accesses into 2 internally locked 16-bit bus cycles; the LOCK# signal is not asserted. The 386 SX Microprocessor breaks unaligned 16-bit or 32-bit data or I/O accesses into 2 or 3 internally locked 16-bit bus cycles. Again, the LOCK# signal is not asserted but a HOLD request will not be recognized until the end of the entire transfer.

Wait states affect HOLD latency. The 386 SX Microprocessor will not honor a HOLD request until the end of the current bus operation, no matter

how many wait states are required. Systems with DMA where data transfer is critical must insure that READY# returns sufficiently soon.

COPROCESSOR INTERFACE SIGNALS (PEREQ, BUSY#, ERROR#)

In the following sections are descriptions of signals dedicated to the numeric coprocessor interface. In addition to the data bus, address bus, and bus cycle definition signals, these following signals control communication between the 386 SX Microprocessor and its 387™ SX processor extension.

Coprocessor Request (PEREQ)

When asserted (HIGH), this input signal indicates a coprocessor request for a data operand to be transferred to/from memory by the 386 SX Microprocessor. In response, the 386 SX Microprocessor transfers information between the coprocessor and memory. Because the 386 SX Microprocessor has internally stored the coprocessor opcode being executed, it performs the requested data transfer with the correct direction and memory address.

PEREQ is a level-sensitive active HIGH asynchronous signal. Setup and hold times, t_{29} and t_{30} , relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This signal is provided with a weak internal pull-down resistor of around 20 K-ohms to ground so that it will not float active when left unconnected.

Coprocessor Busy (BUSY#)

When asserted (LOW), this input indicates the coprocessor is still executing an instruction, and is not yet able to accept another. When the 386 SX Microprocessor encounters any coprocessor instruction which operates on the numerics stack (e.g. load, pop, or arithmetic operation), or the WAIT instruction, this input is first automatically sampled until it is seen to be inactive. This sampling of the BUSY# input prevents overrunning the execution of a previous coprocessor instruction.

The FNINIT, FNSTENV, FNSAVE, FNSTSW, FNSTCW and FNCLEX coprocessor instructions are allowed to execute even if BUSY# is active, since these instructions are used for coprocessor initialization and exception-clearing.

BUSY# is an active LOW, level-sensitive asynchronous signal. Setup and hold times, t_{29} and t_{30} , rela-

tive to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This pin is provided with a weak internal pull-up resistor of around 20 K-ohms to Vcc so that it will not float active when left unconnected.

BUSY# serves an additional function. If BUSY# is sampled LOW at the falling edge of RESET, the 386 SX Microprocessor performs an internal self-test (see **Bus Activity During and Following Reset**. If BUSY# is sampled HIGH, no self-test is performed.

Coprocessor Error (ERROR#)

When asserted (LOW), this input signal indicates that the previous coprocessor instruction generated a coprocessor error of a type not masked by the coprocessor's control register. This input is automatically sampled by the 386 SX Microprocessor when a coprocessor instruction is encountered, and if active, the 386 SX Microprocessor generates exception 16 to access the error-handling software.

Several coprocessor instructions, generally those which clear the numeric error flags in the coprocessor or save coprocessor state, do execute without the 386 SX Microprocessor generating exception 16 even if ERROR# is active. These instructions are FNINIT, FNCLEX, FNSTSW, FNSTSWAX, FNSTCW, FNSTENV and FNSAVE.

ERROR# is an active LOW, level-sensitive asynchronous signal. Setup and hold times, t_{29} and t_{30} , relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This pin is provided with a weak internal pull-up resistor of around 20 K-ohms to Vcc so that it will not float active when left unconnected.

INTERRUPT SIGNALS (INTR, NMI, RESET)

The following descriptions cover inputs that can interrupt or suspend execution of the processor's current instruction stream.

Maskable Interrupt Request (INTR)

When asserted, this input indicates a request for interrupt service, which can be masked by the 386 SX CPU Flag Register IF bit. When the 386 SX Microprocessor responds to the INTR input, it performs two interrupt acknowledge bus cycles and, at the end of the second, latches an 8-bit interrupt vector on D7–D0 to identify the source of the interrupt.

INTR is an active HIGH, level-sensitive asynchronous signal. Setup and hold times, t_{27} and t_{28} , relative to the CLK2 signal must be met to guarantee

recognition at a particular clock edge. To assure recognition of an INTR request, INTR should remain active until the first interrupt acknowledge bus cycle begins. INTR is sampled at the beginning of every instruction in the 386 SX Microprocessor's Execution Unit. In order to be recognized at a particular instruction boundary, INTR must be active at least eight CLK2 clock periods before the beginning of the instruction. If recognized, the 386 SX Microprocessor will begin execution of the interrupt.

Non-Maskable Interrupt Request (NMI)

This input indicates a request for interrupt service which cannot be masked by software. The non-maskable interrupt request is always processed according to the pointer or gate in slot 2 of the interrupt table. Because of the fixed NMI slot assignment, no interrupt acknowledge cycles are performed when processing NMI.

NMI is an active HIGH, rising edge-sensitive asynchronous signal. Setup and hold times, t_{27} and t_{28} , relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. To assure recognition of NMI, it must be inactive for at least eight CLK2 periods, and then be active for at least eight CLK2 periods before the beginning of the instruction boundary in the 386 SX Microprocessor's Execution Unit.

Once NMI processing has begun, no additional NMI's are processed until after the next IRET instruction, which is typically the end of the NMI service routine. If NMI is re-asserted prior to that time, however, one rising edge on NMI will be remembered for processing after executing the next IRET instruction.

Interrupt Latency

The time that elapses before an interrupt request is serviced (interrupt latency) varies according to several factors. This delay must be taken into account by the interrupt source. Any of the following factors can affect interrupt latency:

1. If interrupts are masked, an INTR request will not be recognized until interrupts are reenabled.
2. If an NMI is currently being serviced, an incoming NMI request will not be recognized until the 386 SX Microprocessor encounters the IRET instruction.
3. An interrupt request is recognized only on an instruction boundary of the 386 SX Microprocessor's Execution Unit except for the following cases:
 - Repeat string instructions can be interrupted after each iteration.

- If the instruction loads the Stack Segment register, an interrupt is not processed until after the following instruction, which should be an ESP. This allows the entire stack pointer to be loaded without interruption.
- If an instruction sets the interrupt flag (enabling interrupts), an interrupt is not processed until after the next instruction.

The longest latency occurs when the interrupt request arrives while the 386 SX Microprocessor is executing a long instruction such as multiplication, division, or a task-switch in the protected mode.

4. Saving the Flags register and CS:EIP registers.
5. If interrupt service routine requires a task switch, time must be allowed for the task switch.
6. If the interrupt service routine saves registers that are not automatically saved by the 386 SX Microprocessor.

RESET

This input signal suspends any operation in progress and places the 386 SX Microprocessor in a known reset state. The 386 SX Microprocessor is reset by asserting RESET for 15 or more CLK2 periods (80 or more CLK2 periods before requesting self-test). When RESET is active, all other input pins are ignored, and all other bus pins are driven to an idle bus state as shown in Table 5.5. If RESET and HOLD are both active at a point in time, RESET takes priority even if the 386 SX Microprocessor was in a Hold Acknowledge state prior to RESET active.

RESET is an active HIGH, level-sensitive synchronous signal. Setup and hold times, t_{25} and t_{26} , must be met in order to assure proper operation of the 386 SX Microprocessor.

Table 5.5. Pin State (Bus Idle) During Reset

Pin Name	Signal Level During Reset
ADS#	1
D ₁₅ -D ₀	Float
BHE#, BLE#	0
A ₂₃ -A ₁	1
W/R#	0
D/C#	1
M/IO#	0
LOCK#	1
HLDA	0

5.2 Bus Transfer Mechanism

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte and word lengths may be transferred without restrictions on

physical address alignment. Any byte boundary may be used, although two physical bus cycles are performed as required for unaligned operand transfers.

The 386 SX Microprocessor address signals are designed to simplify external system hardware. Higher-order address bits are provided by A₂₃-A₁. BHE# and BLE# provide linear selects for the two bytes of the 16-bit data bus.

Byte Enable outputs BHE# and BLE# are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 5.6.

Table 5.6. Byte Enables and Associated Data and Operand Bytes

Byte Enable Signal	Associated Data Bus Signals	
BLE#	D ₇ -D ₀	(byte 0 — least significant)
BHE#	D ₁₅ -D ₈	(byte 1 — most significant)

Each bus cycle is composed of at least two bus states. Each bus state requires one processor clock period. Additional bus states added to a single bus cycle are called wait states. See section 5.4 **Bus Functional Description**.

5.3 Memory and I/O Spaces

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. As shown in Figure 5.3, physical memory addresses range from 000000H to 0FFFFFFH (16 megabytes) and I/O addresses from 000000H to 00FFFFH (64 kilobytes). Note the I/O addresses used by the automatic I/O cycles for coprocessor communication are 8000F8H to 8000FFH, beyond the address range of programmed I/O, to allow easy generation of a coprocessor chip select signal using the A₂₃ and M/IO# signals.

5.4 Bus Functional Description

The 386 SX Microprocessor has separate, parallel buses for data and address. The data bus is 16-bits in width, and bidirectional. The address bus provides a 24-bit value using 23 signals for the 23 upper-order address bits and 2 Byte Enable signals to directly indicate the active bytes. These buses are interpreted and controlled by several definition signals.

The definition of each bus cycle is given by three signals: M/IO#, W/R# and D/C#. At the same time, a valid address is present on the byte enable signals, BHE# and BLE#, and the other address signals A₂₃-A₁. A status signal, ADS#, indicates

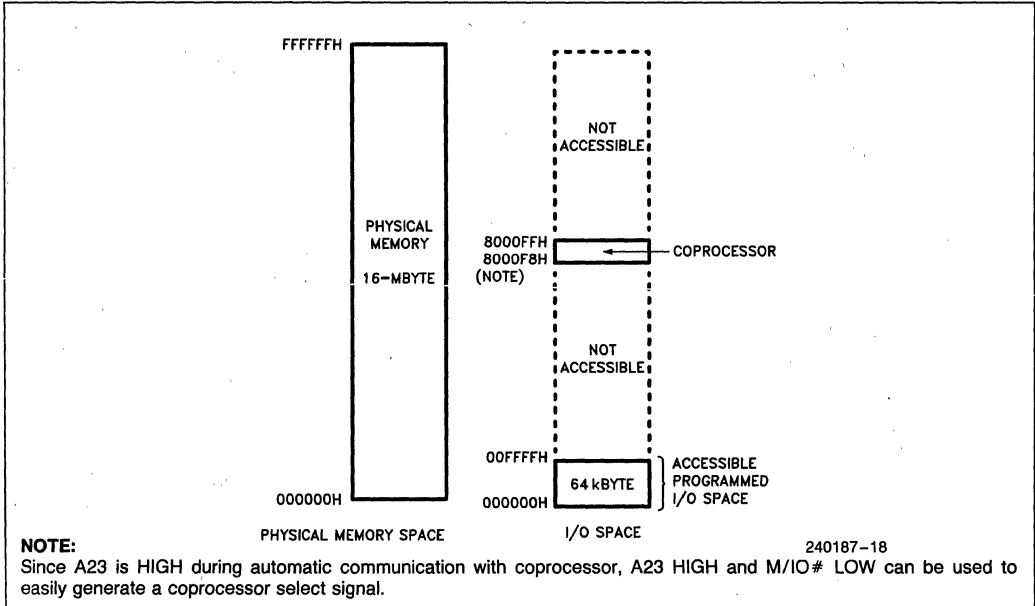


Figure 5.3. Physical Memory and I/O Spaces

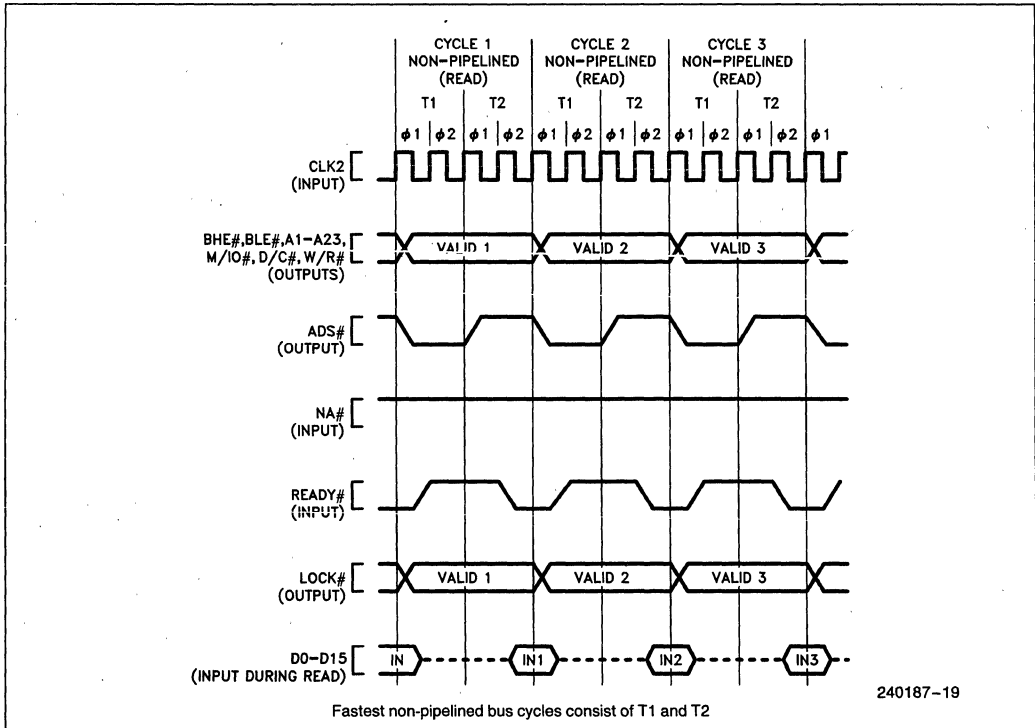


Figure 5.4. Fastest Read Cycles with Non-pipelined Address Timing

when the 386 SX Microprocessor issues a new bus cycle definition and address.

Collectively, the address bus, data bus and all associated control signals are referred to simply as 'the bus'. When active, the bus performs one of the bus cycles below:

1. Read from memory space
2. Locked read from memory space
3. Write to memory space
4. Locked write to memory space
5. Read from I/O space (or coprocessor)
6. Write to I/O space (or coprocessor)
7. Interrupt acknowledge (always locked)
8. Indicate halt, or indicate shutdown

Table 5.2 shows the encoding of the bus cycle definition signals for each bus cycle. See **Bus Cycle Definition Signals** for additional information.

When the 386 SX Microprocessor bus is not performing one of the activities listed above, it is either Idle or in the Hold Acknowledge state, which may be detected externally. The idle state can be identified by the 386 SX Microprocessor giving no further assertions on its address strobe output (ADS#) since the beginning of its most recent bus cycle, and the most recent bus cycle having been terminated. The hold acknowledge state is identified by the 386 SX Microprocessor asserting its hold acknowledge (HLDA) output.

The shortest time unit of bus activity is a bus state. A bus state is one processor clock period (two CLK2 periods) in duration. A complete data transfer occurs during a bus cycle, composed of two or more bus states.

The fastest 386 SX Microprocessor bus cycle requires only two bus states. For example, three consecutive bus read cycles, each consisting of two bus states, are shown by Figure 5.4. The bus states in each cycle are named T1 and T2. Any memory or I/O address may be accessed by such a two-state bus cycle, if the external hardware is fast enough.

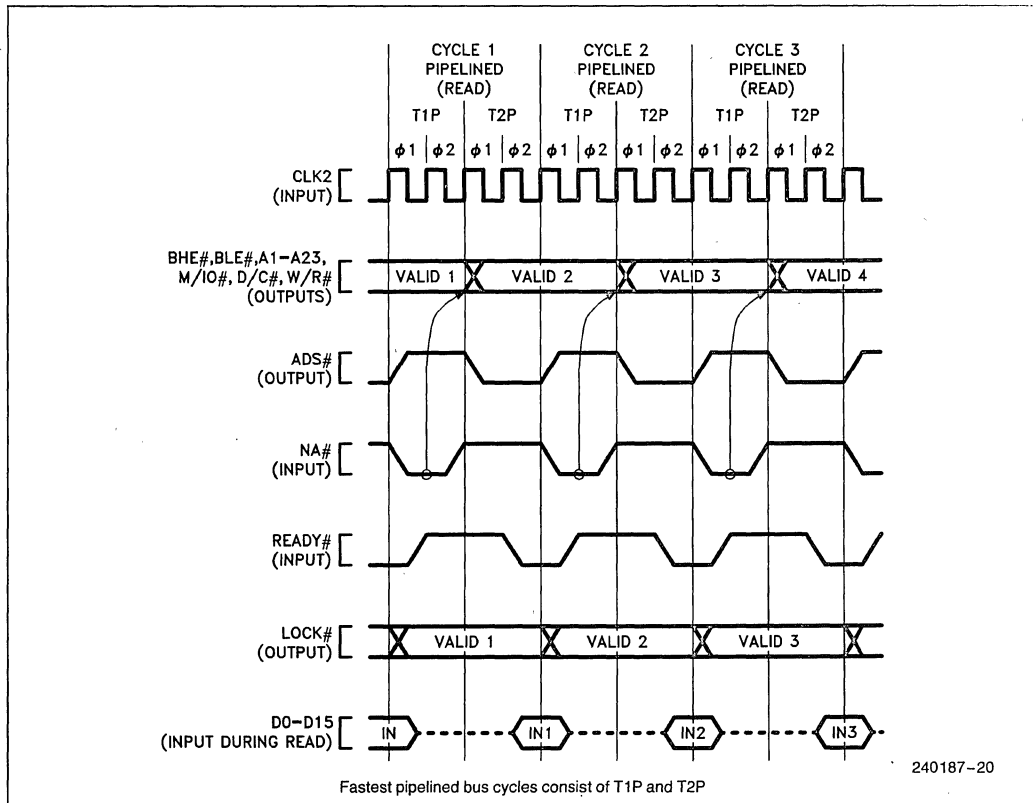


Figure 5.5. Fastest Read Cycles with Pipelined Address Timing

Every bus cycle continues until it is acknowledged by the external system hardware, using the 386 SX Microprocessor **READY#** input. Acknowledging the bus cycle at the end of the first T2 results in the shortest bus cycle, requiring only T1 and T2. If **READY#** is not immediately asserted however, T2 states are repeated indefinitely until the **READY#** input is sampled active.

The address pipelining option provides a choice of bus cycle timings. Pipelined or non-pipelined address timing is selectable on a cycle-by-cycle basis with the Next Address (**NA#**) input.

When address pipelining is selected the address (**BHE#**, **BLE#** and **A₂₃-A₁**) and definition (**W/R#**, **D/C#**, **M/IO#** and **LOCK#**) of the next cycle are available before the end of the current cycle. To signal their availability, the 386 SX Microprocessor ad-

dress status output (**ADS#**) is asserted. Figure 5.5 illustrates the fastest read cycles with pipelined address timing.

Note from Figure 5.5 the fastest bus cycles using pipelined address require only two bus states, named **T1P** and **T2P**. Therefore cycles with pipelined address timing allow the same data bandwidth as non-pipelined cycles, but address-to-data access time is increased by one T-state time compared to that of a non-pipelined cycle.

READ AND WRITE CYCLES

Data transfers occur as a result of bus cycles, classified as read or write cycles. During read cycles, data is transferred from an external device to the processor. During write cycles, data is transferred from the processor to an external device.

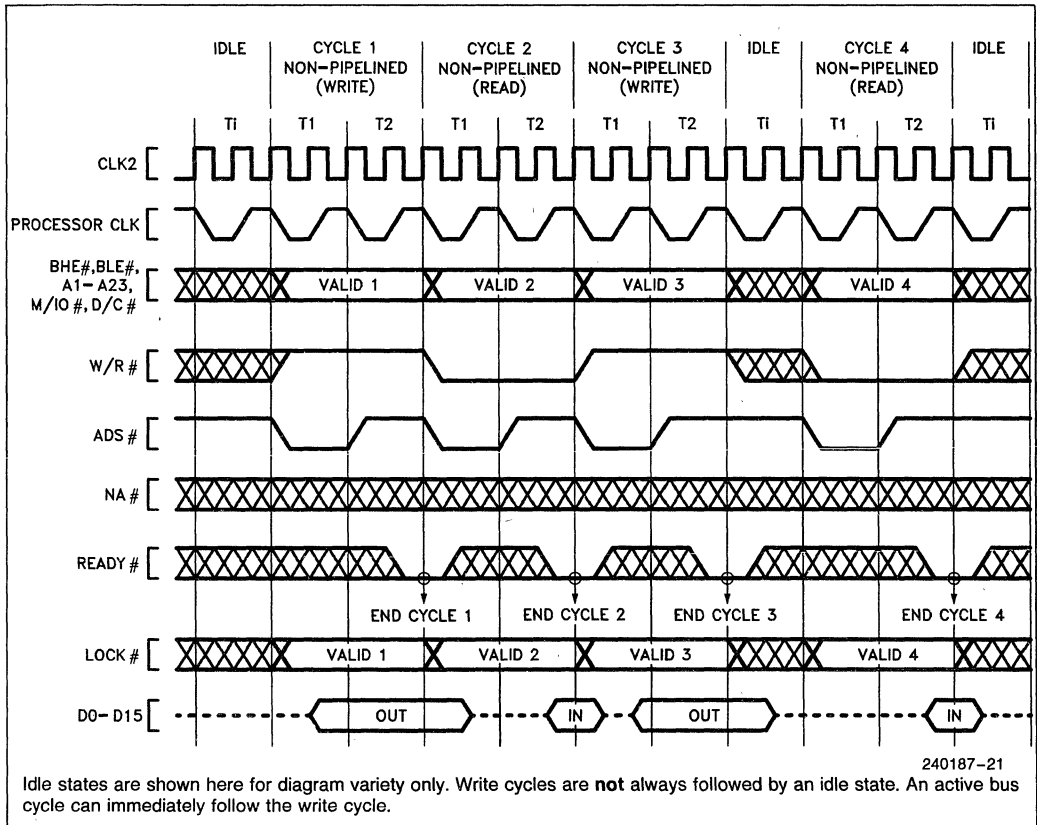


Figure 5.6. Various Bus Cycles with Non-Pipelined Address (zero wait states)

Two choices of address timing are dynamically selectable: non-pipelined or pipelined. After an idle bus state, the processor always uses non-pipelined address timing. However the NA# (Next Address) input may be asserted to select pipelined address timing for the next bus cycle. When pipelining is selected and the 386 SX Microprocessor has a bus request pending internally, the address and definition of the next cycle is made available even before the current bus cycle is acknowledged by READY#.

Terminating a read or write cycle, like any bus cycle, requires acknowledging the cycle by asserting the READY# input. Until acknowledged, the processor inserts wait states into the bus cycle, to allow adjustment for the speed of any external device. External hardware, which has decoded the address and bus cycle type, asserts the READY# input at the appropriate time.

At the end of the second bus state within the bus cycle, READY# is sampled. At that time, if external hardware acknowledges the bus cycle by asserting READY#, the bus cycle terminates as shown in Figure 5.6. If READY# is negated as in Figure 5.7, the 386 SX Microprocessor executes another bus state (a wait state) and READY# is sampled again at the end of that state. This continues indefinitely until the cycle is acknowledged by READY# asserted.

When the current cycle is acknowledged, the 386 SX Microprocessor terminates it. When a read cycle is acknowledged, the 386 SX Microprocessor latches the information present at its data pins. When a write cycle is acknowledged, the 386 SX CPU's write data remains valid throughout phase one of the next bus state, to provide write data hold time.

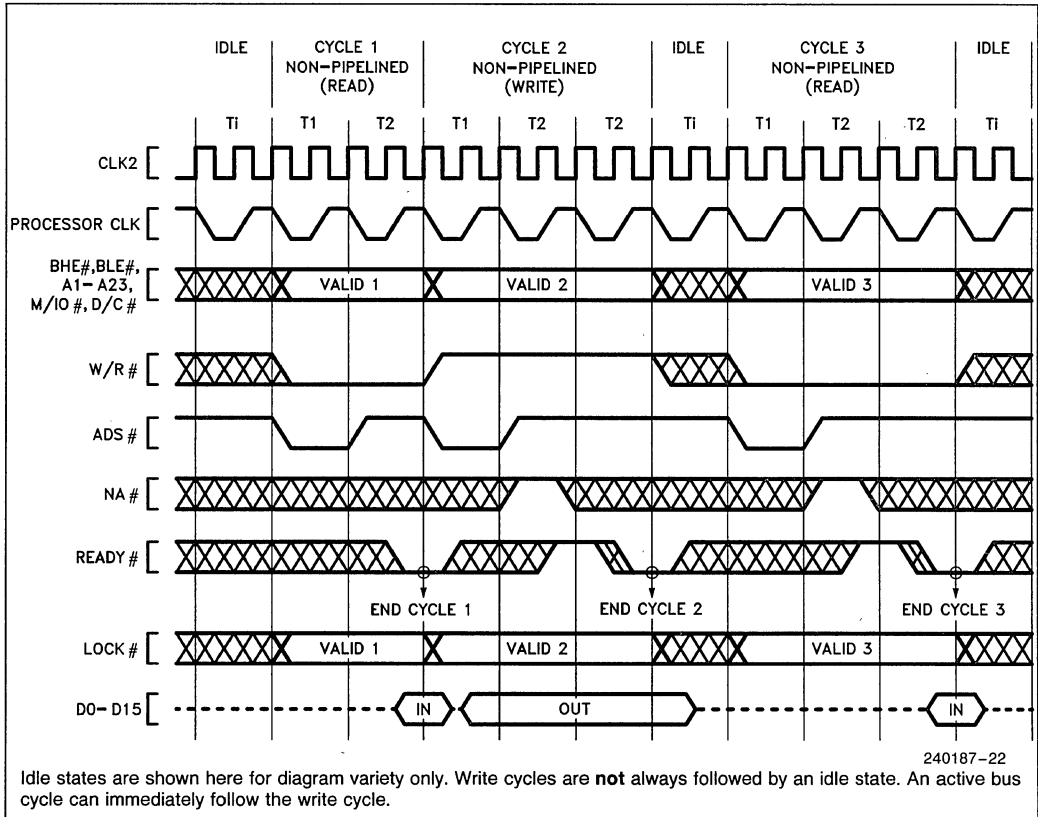


Figure 5.7. Various Bus Cycles with Non-Pipelined Address (various number of wait states)

Non-Pipelined Address

Any bus cycle may be performed with non-pipelined address timing. For example, Figure 5.6 shows a mixture of read and write cycles with non-pipelined address timing. Figure 5.6 shows that the fastest possible cycles with non-pipelined address have two bus states per bus cycle. The states are named T1 and T2. In phase one of T1, the address signals and bus cycle definition signals are driven valid and, to signal their availability, address strobe (ADS#) is simultaneously asserted.

During read or write cycles, the data bus behaves as follows. If the cycle is a read, the 386 SX Microprocessor floats its data signals to allow driving by the external device being addressed. **The 386 SX Microprocessor requires that all data bus pins be at a valid logic state (HIGH or LOW) at the end of each read cycle, when READY# is asserted. The system MUST be designed to meet this requirement.** If the cycle is a write, data signals are driven by the 386 SX Microprocessor beginning in phase two of T1 until phase one of the bus state following cycle acknowledgment.

Figure 5.7 illustrates non-pipelined bus cycles with one wait state added to Cycles 2 and 3. READY# is sampled inactive at the end of the first T2 in Cycles 2 and 3. Therefore Cycles 2 and 3 have T2 repeated again. At the end of the second T2, READY# is sampled active.

When address pipelining is not used, the address and bus cycle definition remain valid during all wait states. When wait states are added and it is desirable to maintain non-pipelined address timing, it is necessary to negate NA# during each T2 state except the last one, as shown in Figure 5.7 Cycles 2 and 3. If NA# is sampled active during a T2 other than the last one, the next state would be T2I or T2P instead of another T2.

When address pipelining is not used, the bus states and transitions are completely illustrated by Figure 5.8. The bus transitions between four possible states, T1, T2, Ti, and Th. Bus cycles consist of T1 and T2, with T2 being repeated for wait states. Otherwise the bus may be idle, Ti, or in the hold acknowledgment state Th.

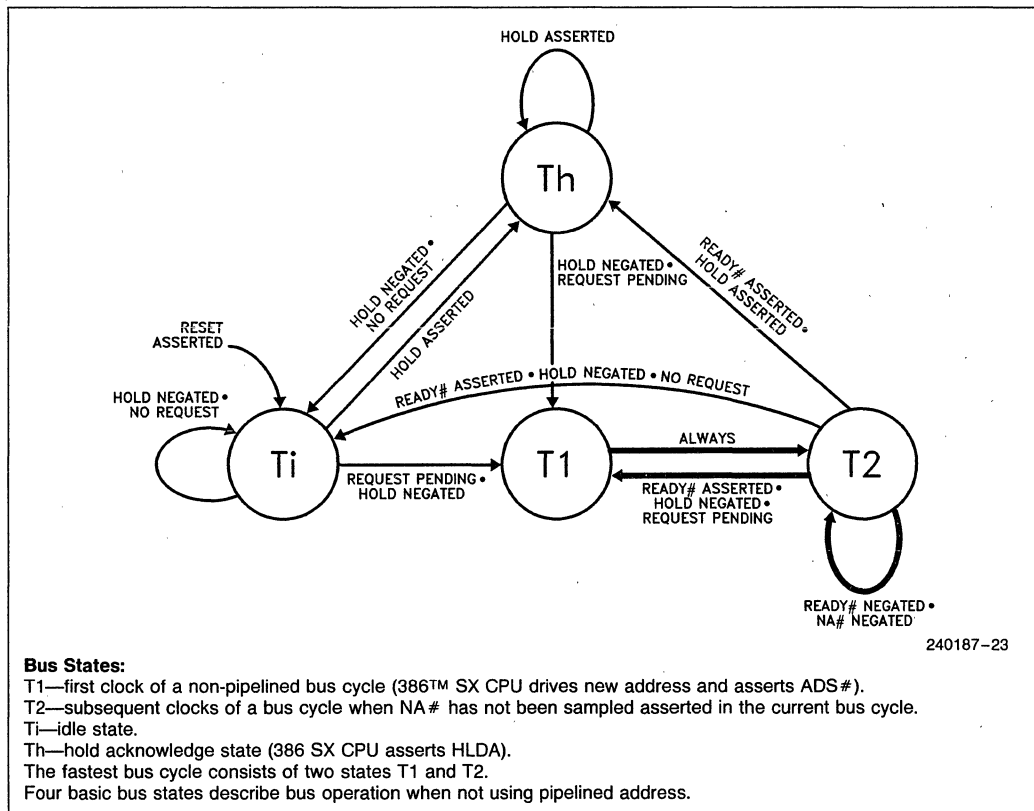


Figure 5.8. Bus States (not using pipelined address)

Bus cycles always begin with T1. T1 always leads to T2. If a bus cycle is not acknowledged during T2 and NA# is inactive, T2 is repeated. When a cycle is acknowledged during T2, the following state will be T1 of the next bus cycle if a bus request is pending internally, or T_i if there is no bus request pending, or T_h if the HOLD input is being asserted.

Use of pipelined address allows the 386 SX Microprocessor to enter three additional bus states not shown in Figure 5.8. Figure 5.12 is the complete bus state diagram, including pipelined address cycles.

Pipelined Address

Address pipelining is the option of requesting the address and the bus cycle definition of the next in-

ternally pending bus cycle before the current bus cycle is acknowledged with READY# asserted. ADS# is asserted by the 386 SX Microprocessor when the next address is issued. The address pipelining option is controlled on a cycle-by-cycle basis with the NA# input signal.

Once a bus cycle is in progress and the current address has been valid for at least one entire bus state, the NA# input is sampled at the end of every phase one until the bus cycle is acknowledged. During non-pipelined bus cycles NA# is sampled at the end of phase one in every T2. An example is Cycle 2 in Figure 5.9, during which NA# is sampled at the end of phase one of every T2 (it was asserted once during the first T2 and has no further effect during that bus cycle).

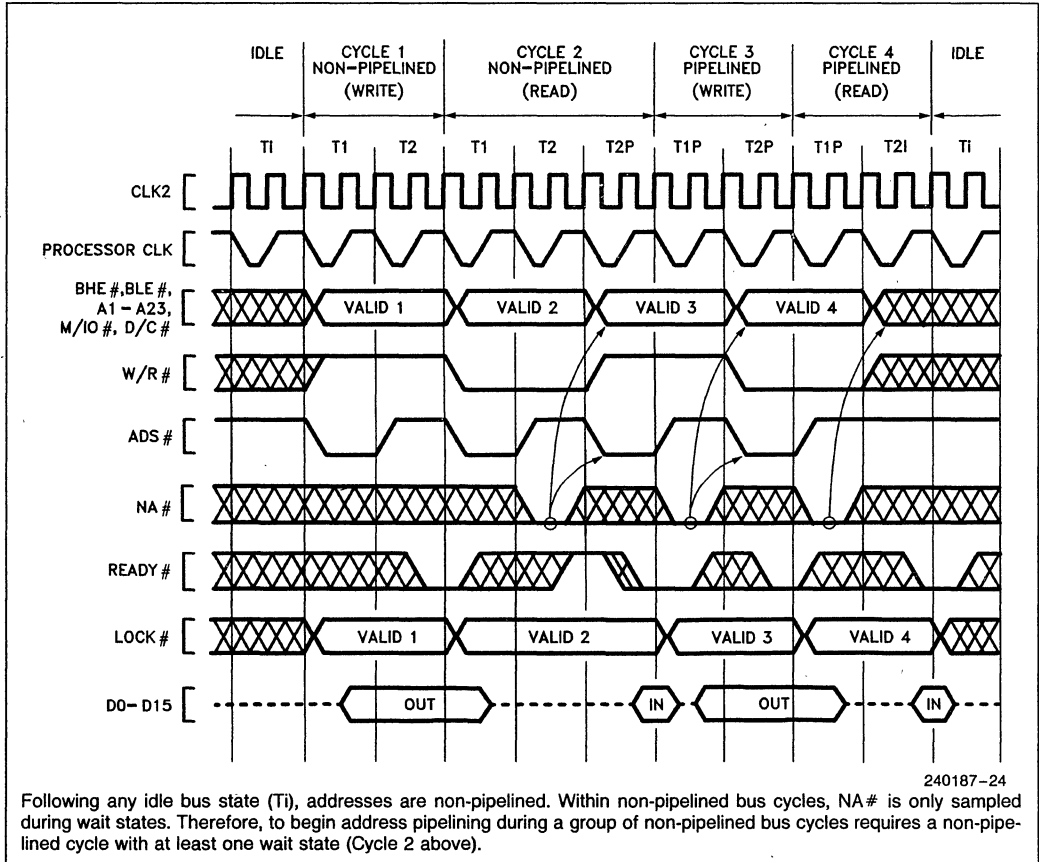


Figure 5.9. Transitioning to Pipelined Address During Burst of Bus Cycles

If NA# is sampled active, the 386 SX Microprocessor is free to drive the address and bus cycle definition of the next bus cycle, and assert ADS#, as soon as it has a bus request internally pending. It may drive the next address as early as the next bus state, whether the current bus cycle is acknowledged at that time or not.

Regarding the details of address pipelining, the 386 SX Microprocessor has the following characteristics:

1. The next address may appear as early as the bus state after NA# was sampled active (see Figures 5.9 or 5.10). In that case, state T2P is entered immediately. However, when there is not an internal bus request already pending, the next address will not be available immediately after NA# is asserted and T2I is entered instead of T2P (see Fig-

ure 5.11 Cycle 3). Provided the current bus cycle isn't yet acknowledged by READY# asserted, T2P will be entered as soon as the 386 SX Microprocessor does drive the next address. External hardware should therefore observe the ADS# output as confirmation the next address is actually being driven on the bus.

2. Any address which is validated by a pulse on the ADS# output will remain stable on the address pins for at least two processor clock periods. The 386 SX Microprocessor cannot produce a new address more frequently than every two processor clock periods (see Figures 5.9, 5.10, and 5.11).
3. Only the address and bus cycle definition of the very next bus cycle is available. The pipelining capability cannot look further than one bus cycle ahead (see Figure 5.11 Cycle 1).

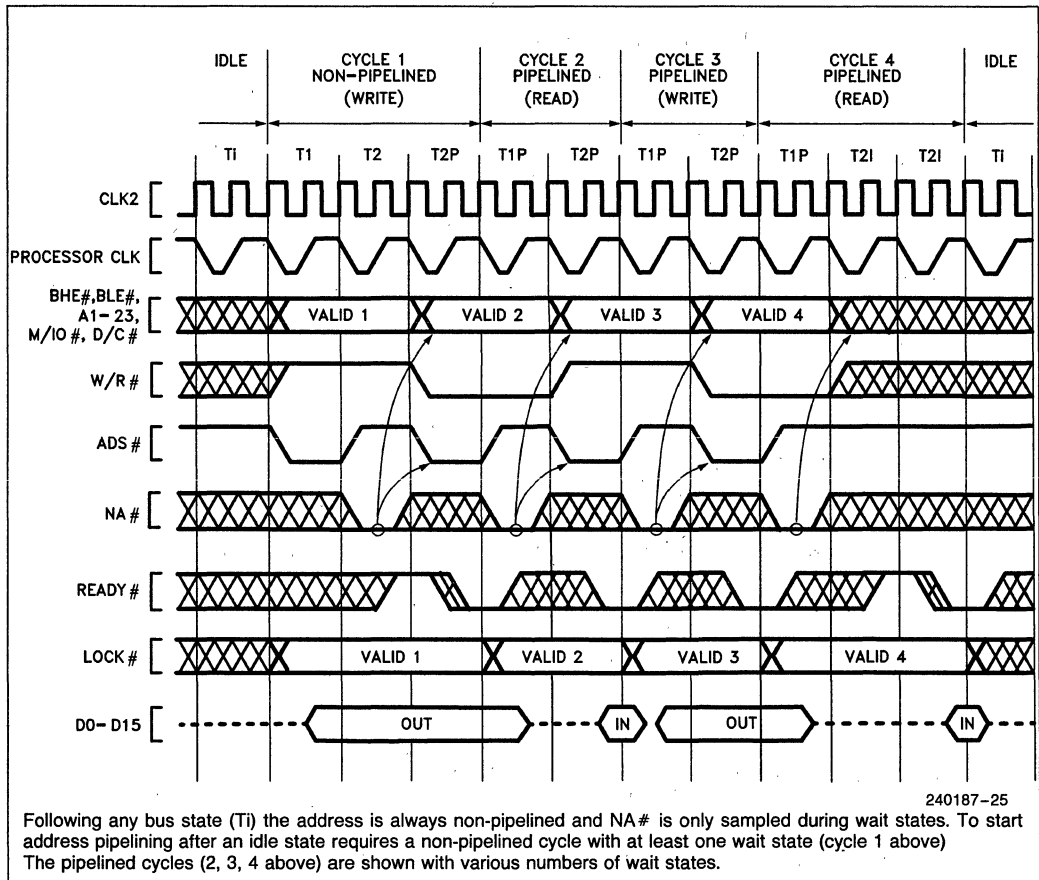


Figure 5.10. Fastest Transition to Pipelined Address Following Idle Bus State

The complete bus state transition diagram, including operation with pipelined address is given by Figure 5.12. Note it is a superset of the diagram for non-pipelined address only, and the three additional bus states for pipelined address are drawn in bold.

The fastest bus cycle with pipelined address consists of just two bus states, T1P and T2P (recall for non-pipelined address it is T1 and T2). T1P is the first bus state of a pipelined cycle.

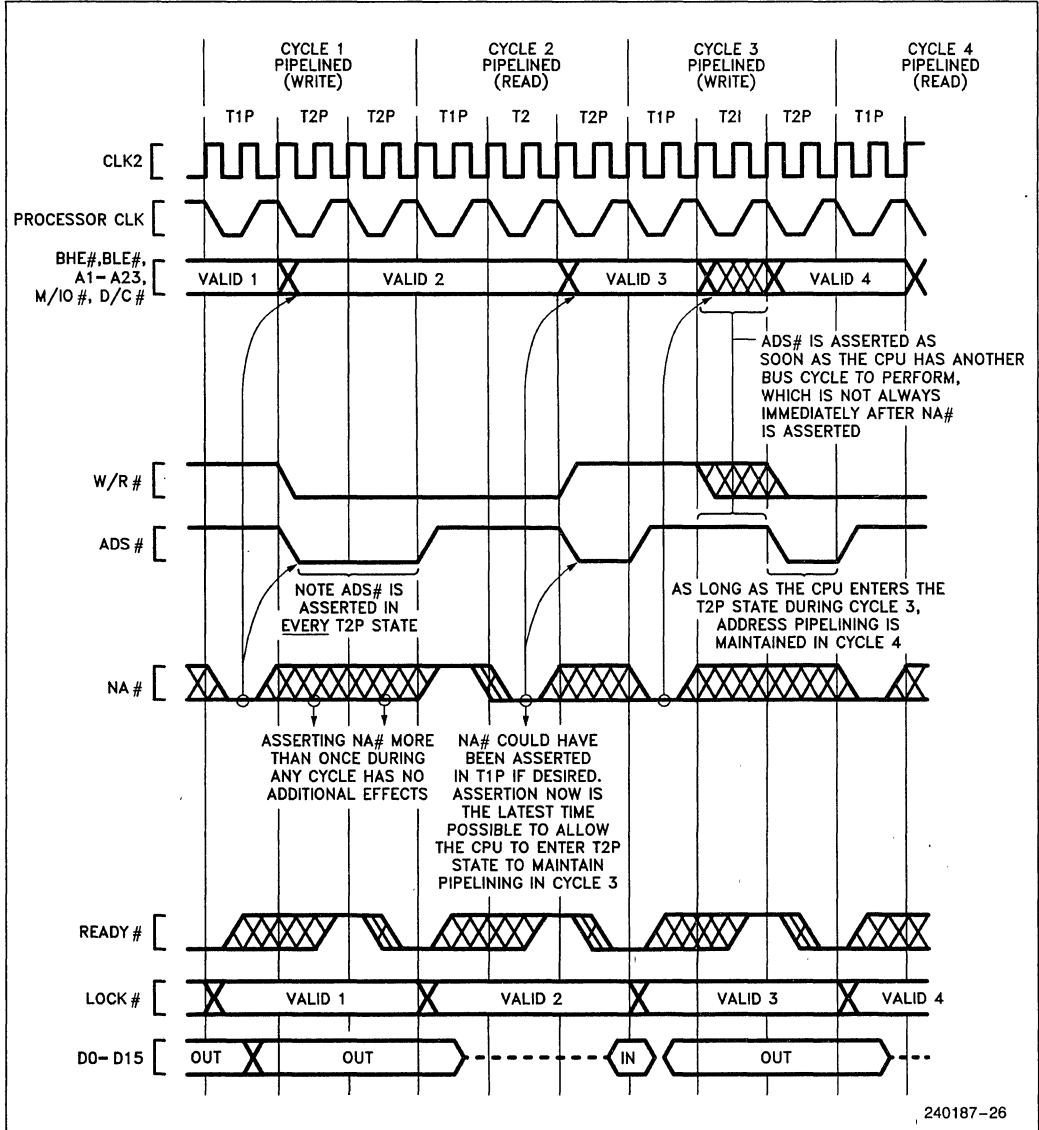


Figure 5.11. Details of Address Pipelining During Cycles with Wait States

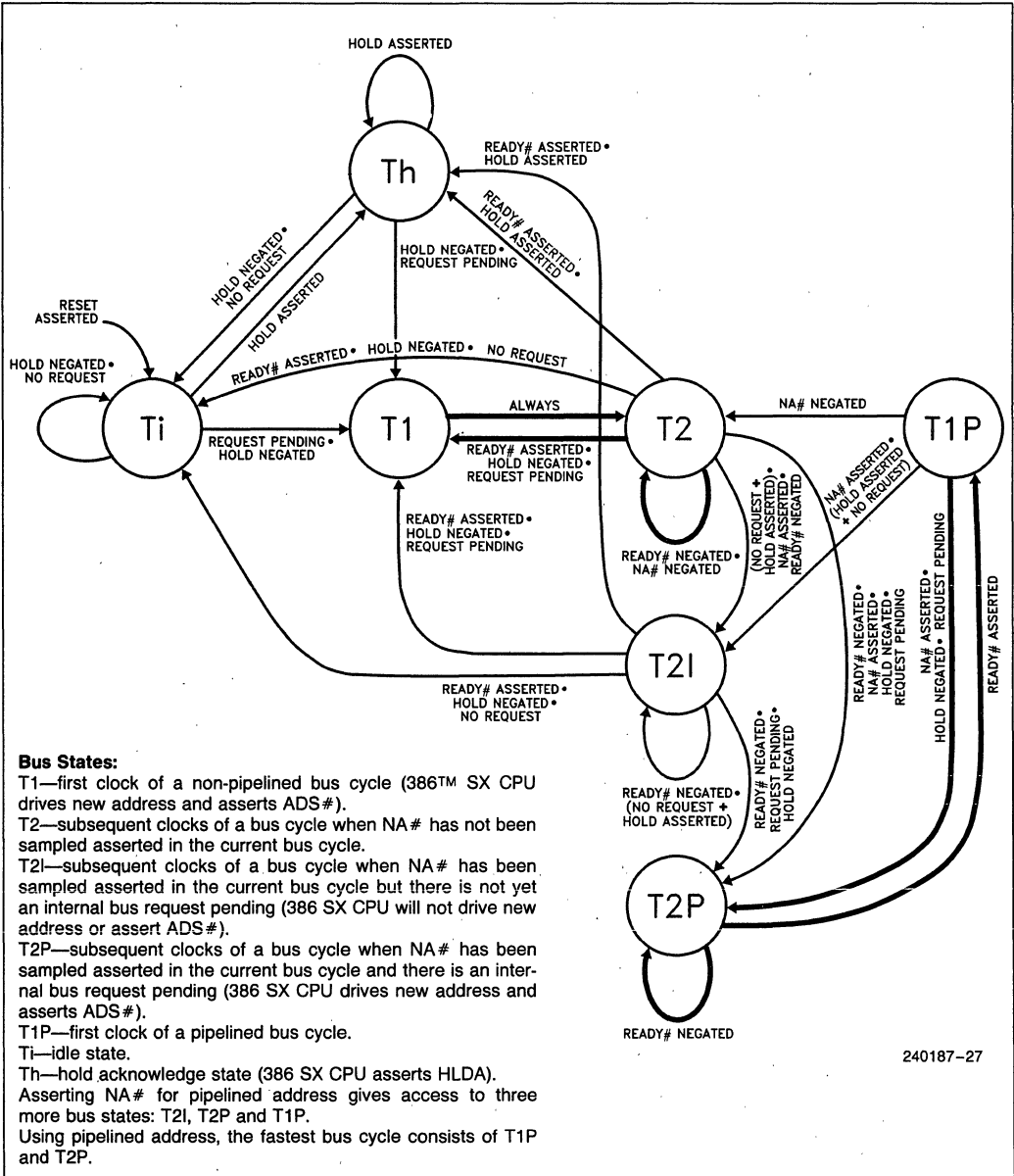


Figure 5.12. Complete Bus States (including pipelined address)

Initiating and Maintaining Pipelined Address

Using the state diagram Figure 5.12, observe the transitions from an idle state, T_i , to the beginning of a pipelined bus cycle T1P. From an idle state, T_i , the first bus cycle must begin with T1, and is therefore a non-pipelined bus cycle. The next bus cycle will be pipelined, however, provided $NA\#$ is asserted and the first bus cycle ends in a T2P state (the address for the next bus cycle is driven during T2P). The fastest path from an idle state to a bus cycle with pipelined address is shown in bold below:

T_i, T_i, T_i	$T1 - T2 - T2P$	$T1P - T2P$
idle	non-pipelined	pipelined
states	cycle	cycle

T1-T2-T2P are the states of the bus cycle that establish address pipelining for the next bus cycle, which begins with T1P. The same is true after a bus hold state, shown below:

T_h, T_h, T_h	$T1 - T2 - T2P$	$T1P - T2P$
hold acknowledge	non-pipelined	pipelined
states	cycle	cycle

The transition to pipelined address is shown functionally by Figure 5.10 Cycle 1. Note that Cycle 1 is used to transition into pipelined address timing for the subsequent Cycles 2, 3 and 4, which are pipelined. The $NA\#$ input is asserted at the appropriate time to select address pipelining for Cycles 2, 3 and 4.

Once a bus cycle is in progress and the current address has been valid for one entire bus state, the $NA\#$ input is sampled at the end of every phase one until the bus cycle is acknowledged. Sampling begins in T2 during Cycle 1 in Figure 5.10. Once $NA\#$ is sampled active during the current cycle, the 386 SX Microprocessor is free to drive a new address and bus cycle definition on the bus as early as the next bus state. In Figure 5.10 Cycle 1 for example, the next address is driven during state T2P. Thus Cycle 1 makes the transition to pipelined address timing, since it begins with T1 but ends with T2P. Because the address for Cycle 2 is available before Cycle 2 begins, Cycle 2 is called a pipelined

bus cycle, and it begins with T1P. Cycle 2 begins as soon as $READY\#$ asserted terminates Cycle 1.

Examples of transition bus cycles are Figure 5.10 Cycle 1 and Figure 5.9 Cycle 2. Figure 5.10 shows transition during the very first cycle after an idle bus state, which is the fastest possible transition into address pipelining. Figure 5.9 Cycle 2 shows a transition cycle occurring during a burst of bus cycles. In any case, a transition cycle is the same whenever it occurs: it consists at least of T1, T2 ($NA\#$ is asserted at that time), and T2P (provided the 386 SX Microprocessor has an internal bus request already pending, which it almost always has). T2P states are repeated if wait states are added to the cycle.

Note that only three states (T1, T2 and T2P) are required in a bus cycle performing a **transition** from non-pipelined address into pipelined address timing, for example Figure 5.10 Cycle 1. Figure 5.10 Cycles 2, 3 and 4 show that address pipelining can be maintained with two-state bus cycles consisting only of T1P and T2P.

Once a pipelined bus cycle is in progress, pipelined timing is maintained for the next cycle by asserting $NA\#$ and detecting that the 386 SX Microprocessor enters T2P during the current bus cycle. The current bus cycle must end in state T2P for pipelining to be maintained in the next cycle. T2P is identified by the assertion of $ADS\#$. Figures 5.9 and 5.10 however, each show pipelining ending after Cycle 4 because Cycle 4 ends in T2I. This indicates the 386 SX Microprocessor didn't have an internal bus request prior to the acknowledgement of Cycle 4. If a cycle ends with a T2 or T2I, the next cycle will not be pipelined.

Realistically, address pipelining is almost always maintained as long as $NA\#$ is sampled asserted. This is so because in the absence of any other request, a code prefetch request is always internally pending until the instruction decoder and code prefetch queue are completely full. Therefore, address pipelining is maintained for long bursts of bus cycles, if the bus is available (i.e., $HOLD$ inactive) and $NA\#$ is sampled active in each of the bus cycles.

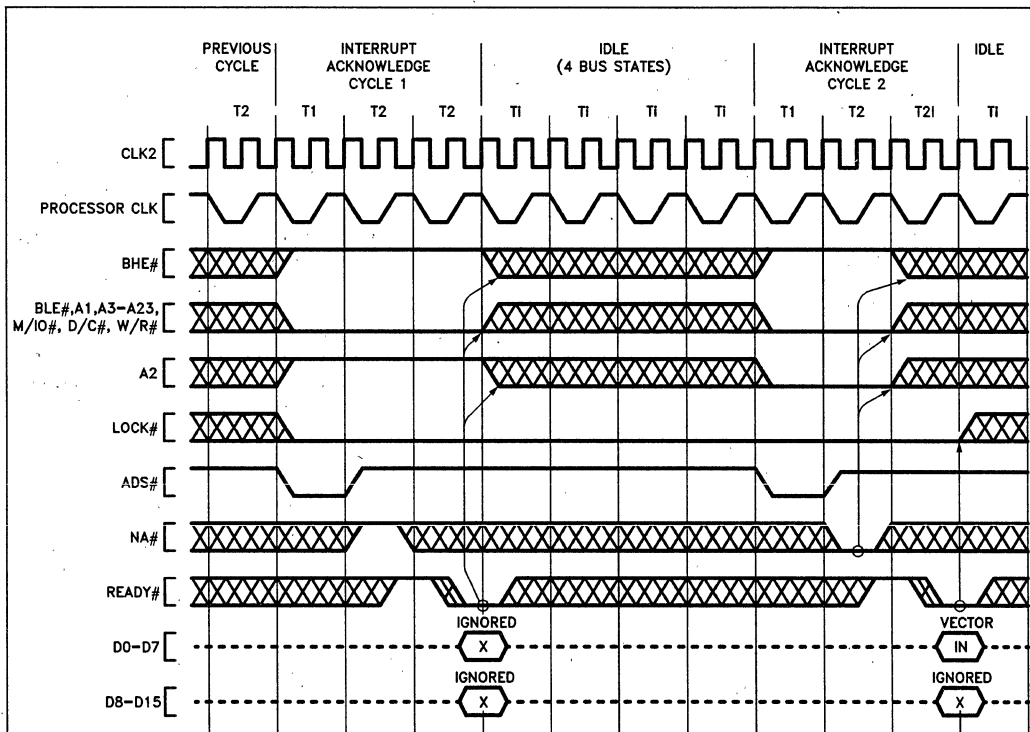
INTERRUPT ACKNOWLEDGE (INTA) CYCLES

In response to an interrupt request on the INTR input when interrupts are enabled, the 386 SX Microprocessor performs two interrupt acknowledge cycles. These bus cycles are similar to read cycles in that bus definition signals define the type of bus activity taking place, and each cycle continues until acknowledged by READY# sampled active.

The state of A₂ distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A₂₃-A₃, A₁, BLE# LOW, A₂ and BHE# HIGH). The byte address driven during the second interrupt acknowledge cycle is 0 (A₂₃-A₁, BLE# LOW, and BHE# HIGH).

The LOCK# output is asserted from the beginning of the first interrupt acknowledge cycle until the end of the second interrupt acknowledge cycle. Four idle bus states, T_i, are inserted by the 386 SX Microprocessor between the two interrupt acknowledge cycles for compatibility with spec TRHRL of the 8259A Interrupt Controller.

During both interrupt acknowledge cycles, D₁₅-D₀ float. No data is read at the end of the first interrupt acknowledge cycle. At the end of the second interrupt acknowledge cycle, the 386 SX Microprocessor will read an external interrupt vector from D₇-D₀ of the data bus. The vector indicates the specific interrupt number (from 0-255) requiring service.



240187-28

Interrupt Vector (0-255) is read on D0-D7 at end of second interrupt Acknowledge bus cycle. Because each Interrupt Acknowledge bus cycle is followed by idle bus states, asserting NA# has no practical effect. Choose the approach which is simplest for your system hardware design.

Figure 5.13. Interrupt Acknowledge Cycles

HALT INDICATION CYCLE

The execution unit halts as a result of executing a HLT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the state of the bus

definition signals shown on page 40, **Bus Cycle Definition Signals**, and an address of 2. The halt indication cycle must be acknowledged by **READY#** asserted. A halted 386 SX Microprocessor resumes execution when **INTR** (if interrupts are enabled), **NMI** or **RESET** is asserted.

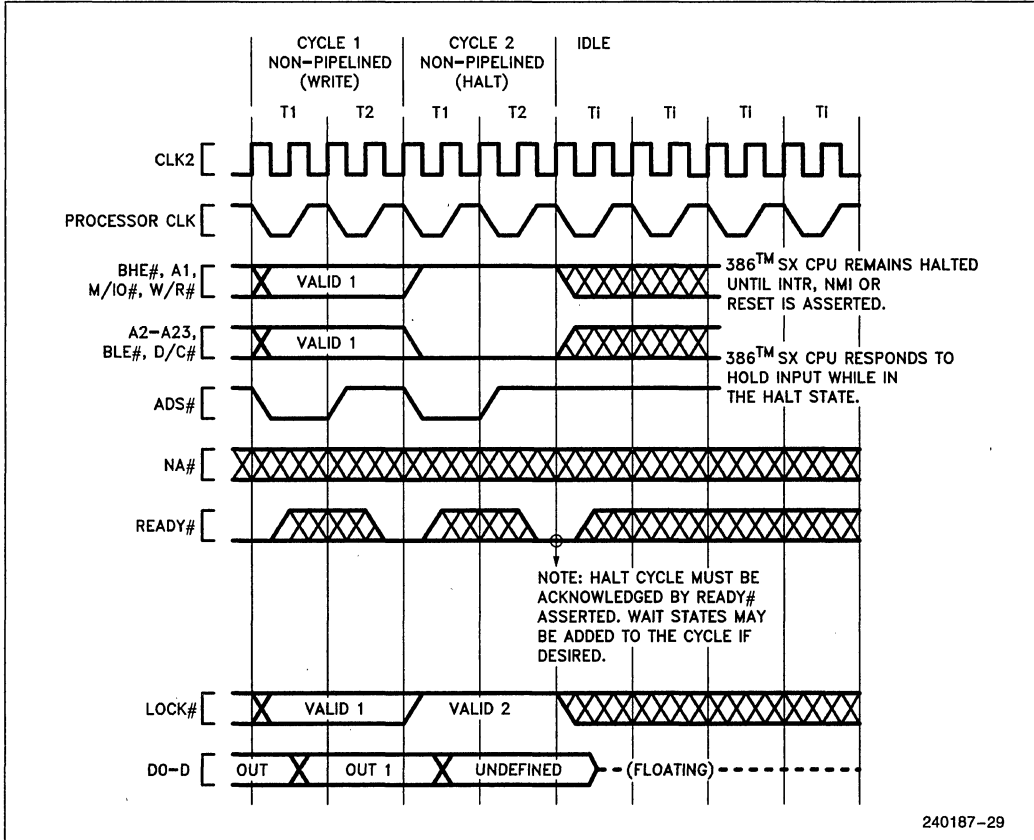


Figure 5.14. Example Halt Indication Cycle from Non-Pipelined Cycle

SHUTDOWN INDICATION CYCLE

The 386 SX Microprocessor shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the state of the bus definition signals shown in **Bus Cycle Definition Signals** and an address of 0. The shutdown indication cycle must be acknowledged by **READY#** asserted. A shutdown 386 SX Microprocessor resumes execution when **NMI** or **RESET** is asserted.

ENTERING AND EXITING HOLD ACKNOWLEDGE

The bus hold acknowledge state, T_h , is entered in response to the **HOLD** input being asserted. In the bus hold acknowledge state, the 386 SX Microprocessor floats all outputs or bidirectional signals, except for **HLDA**. **HLDA** is asserted as long as the 386 SX Microprocessor remains in the bus hold acknowledge state. In the bus hold acknowledge state, all inputs except **HOLD** and **RESET** are ignored.

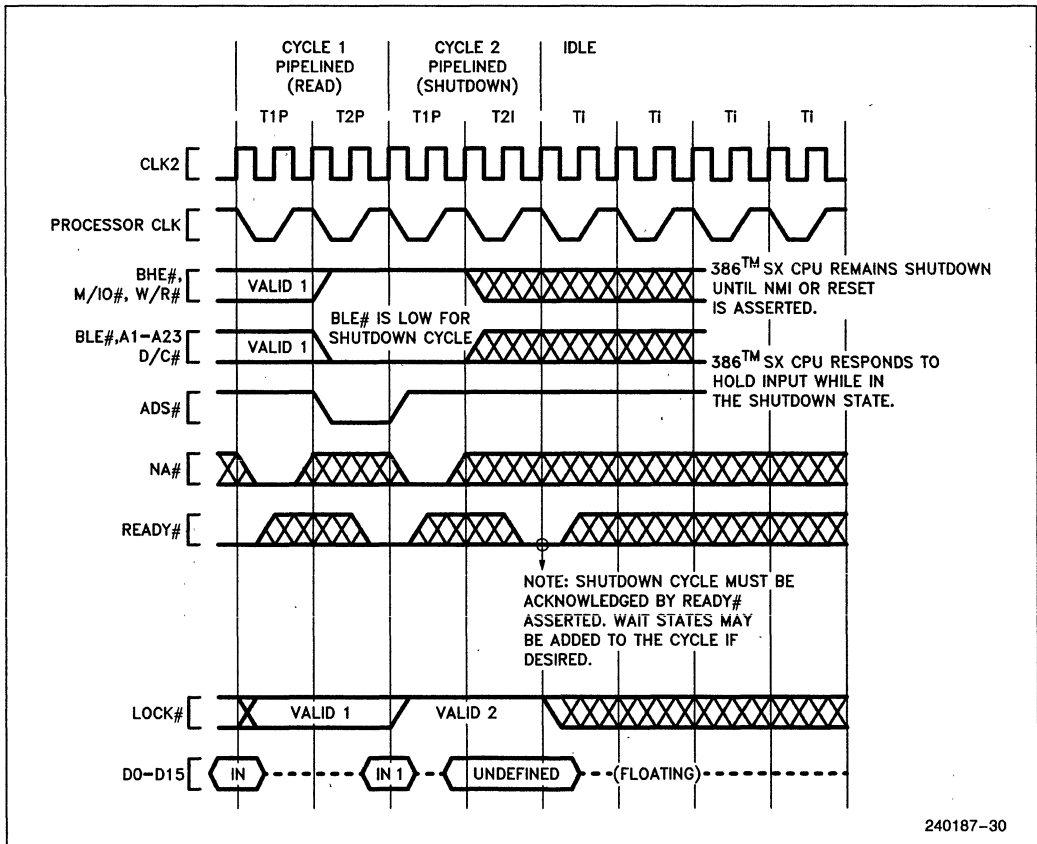


Figure 5.15. Example Shutdown Indication Cycle from Non-Pipelined Cycle

T_h may be entered from a bus idle state as in Figure 5.16 or after the acknowledgement of the current physical bus cycle if the LOCK# signal is not asserted, as in Figures 5.17 and 5.18.

T_h is exited in response to the HOLD input being negated. The following state will be T_i as in Figure 5.16 if no bus request is pending. The following bus state will be T_1 if a bus request is internally pending, as in Figures 5.17 and 5.18. T_h is exited in response to RESET being asserted.

If a rising edge occurs on the edge-triggered NMI input while in T_h , the event is remembered as a non-maskable interrupt 2 and is serviced when T_h is exited unless the 386 SX Microprocessor is reset before T_h is exited.

RESET DURING HOLD ACKNOWLEDGE

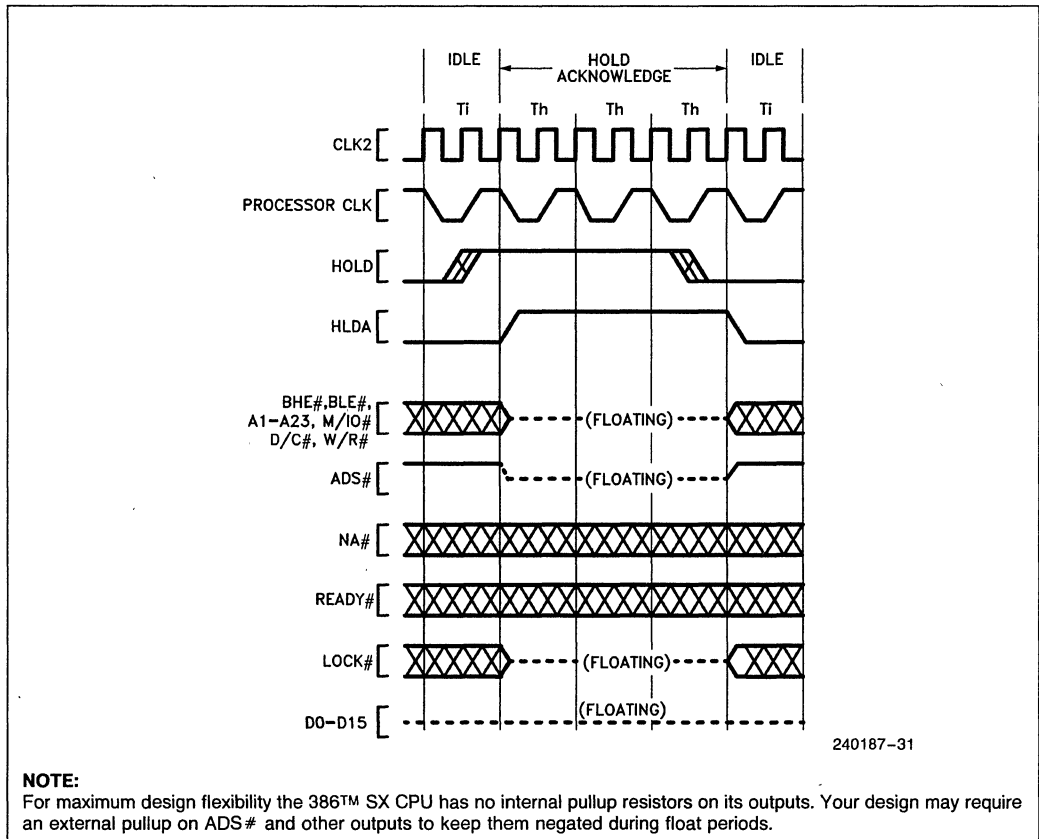
RESET being asserted takes priority over HOLD being asserted. If RESET is asserted while HOLD re-

mains asserted, the 386 SX Microprocessor drives its pins to defined states during reset, as in **Table 5.5 Pin State During Reset**, and performs internal reset activity as usual.

If HOLD remains asserted when RESET is inactive, the 386 SX Microprocessor enters the hold acknowledge state before performing its first bus cycle, provided HOLD is still asserted when the 386 SX Microprocessor would otherwise perform its first bus cycle.

BUS ACTIVITY DURING AND FOLLOWING RESET

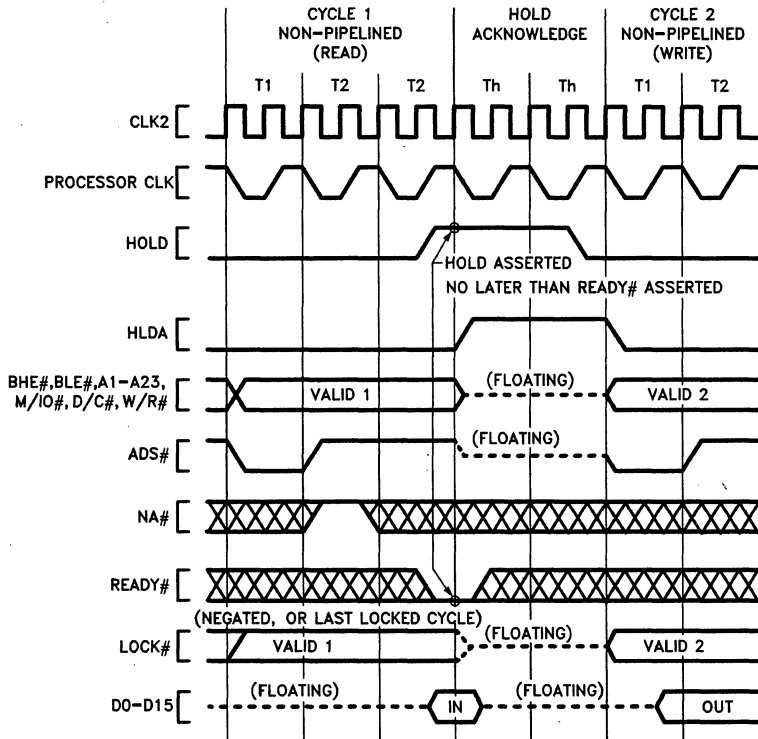
RESET is the highest priority input signal, capable of interrupting any processor activity when it is asserted. A bus cycle in progress can be aborted at any stage, or idle states or bus hold acknowledge states discontinued so that the reset state is established.



NOTE:

For maximum design flexibility the 386™ SX CPU has no internal pullup resistors on its outputs. Your design may require an external pullup on ADS# and other outputs to keep them negated during float periods.

Figure 5.16. Requesting Hold from Idle Bus



240187-32

NOTE:

HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t_{23} and t_{24}) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

Figure 5.17. Requesting Hold from Active Bus (NA# inactive)

RESET should remain asserted for at least 15 CLK2 periods to ensure it is recognized throughout the 386 SX Microprocessor, and at least 80 CLK2 periods if self-test is going to be requested at the falling edge. RESET asserted pulses less than 15 CLK2 periods may not be recognized. RESET pulses less than 80 CLK2 periods followed by a self-test may cause the self-test to report a failure when no true failure exists.

Provided the RESET falling edge meets setup and hold times t_{25} and t_{26} , the internal processor clock phase is defined at that time as illustrated by Figure 5.19 and Figure 7.7.

A self-test may be requested at the time RESET goes inactive by having the BUSY# input at a LOW level as shown in Figure 5.19. The self-test requires approximately $(2^{20} + 60)$ CLK2 periods to complete. The self-test duration is not affected by the test results. Even if the self-test indicates a problem, the 386 SX Microprocessor attempts to proceed with the reset sequence afterwards.

After the RESET falling edge (and after the self-test if it was requested) the 386 SX Microprocessor performs an internal initialization sequence for approximately 350 to 450 CLK2 periods.

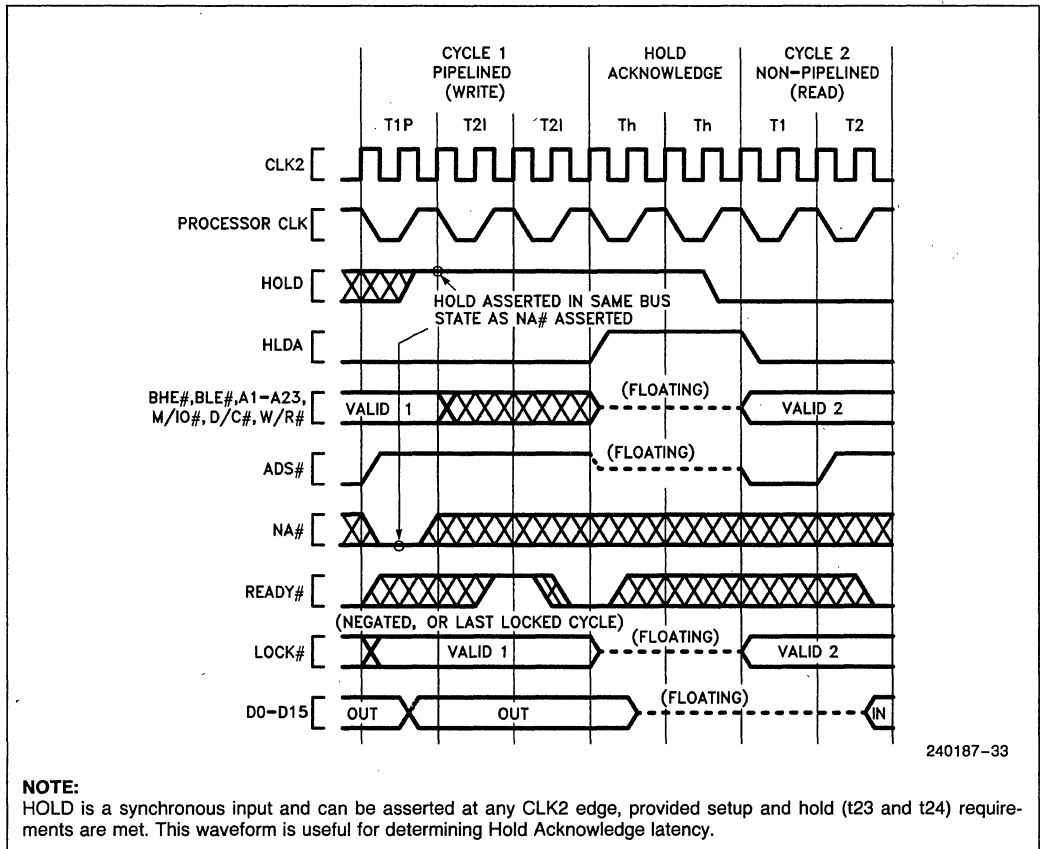
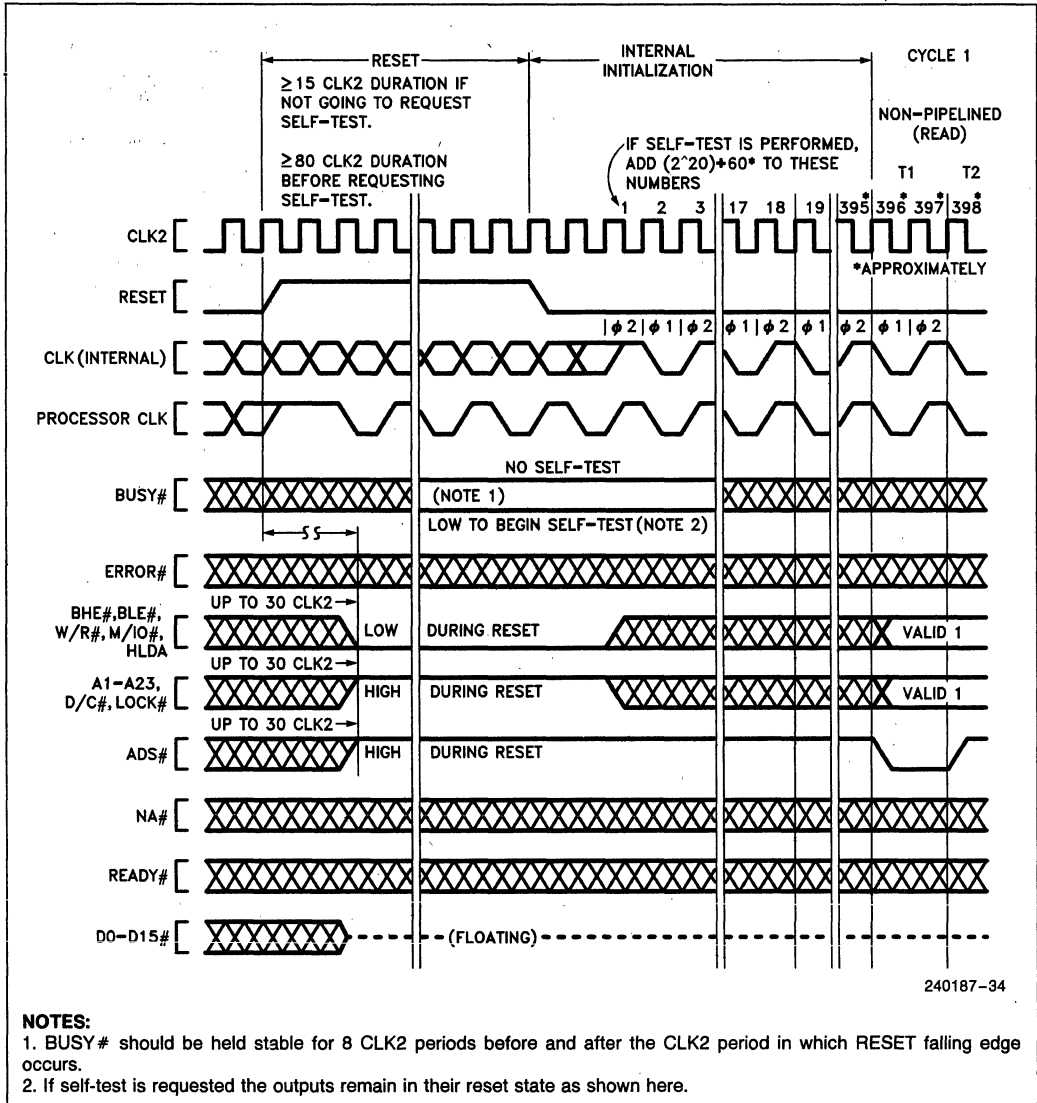


Figure 5.18. Requesting Hold from Idle Bus (NA# active)


Figure 5.19. Bus Activity from Reset Until First Code Fetch

5.5 Self-test Signature

Upon completion of self-test (if self-test was requested by driving **BUSY#** LOW at the falling edge of **RESET**) the **EAX** register will contain a signature of 00000000H indicating the 386 SX Microprocessor passed its self-test of microcode and major PLA contents with no problems detected. The passing signature in **EAX**, 00000000H, applies to all revision levels. Any non-zero signature indicates the unit is faulty.

5.6 Component and Revision Identifiers

To assist users, the 386 SX Microprocessor after reset holds a component identifier and revision identifier in its **DX** register. The upper 8 bits of **DX** hold 23H as identification of the 386 SX Microprocessor (the lower nibble, 03H, refers to the Intel386 DX Architecture. The upper nibble, 02H, refers to the second member of the Intel386 DX Family). The lower 8 bits of **DX** hold an 8-bit unsigned binary number related to the component revision level. The revision identifier will, in general, chronologically track those component steppings which are intended to have certain improvements or distinction from previous steppings. The 386 SX Microprocessor revision identifier will track that of the 386 DX CPU where possible.

The revision identifier is intended to assist users to a practical extent. However, the revision identifier value is not guaranteed to change with every stepping revision, or to follow a completely uniform numerical sequence, depending on the type or intention of revision, or manufacturing materials required to be changed. Intel has sole discretion over these characteristics of the component.

Table 5.7. Component and Revision Identifier History

Stepping	Revision Identifier
A0	04H
B	05H
C	08H

5.7 Coprocessor Interfacing

The 386 SX Microprocessor provides an automatic interface for the Intel 387 SX numeric floating-point coprocessor. The 387 SX coprocessor uses an I/O mapped interface driven automatically by the 386 SX Microprocessor and assisted by three dedicated signals: **BUSY#**, **ERROR#** and **PEREQ**.

As the 386 SX Microprocessor begins supporting a coprocessor instruction, it tests the **BUSY#** and **ERROR#** signals to determine if the coprocessor can accept its next instruction. Thus, the **BUSY#** and **ERROR#** inputs eliminate the need for any 'pre-

amble' bus cycles for communication between processor and coprocessor. The 387™ SX can be given its command opcode immediately. The dedicated signals provide instruction synchronization, and eliminate the need of using the **WAIT** opcode (9BH) for 387™ SX instruction synchronization (the **WAIT** opcode was required when the 8086 or 8088 was used with the 8087 coprocessor).

Custom coprocessors can be included in 386 SX Microprocessor based systems by memory-mapped or I/O-mapped interfaces. Such coprocessor interfaces allow a completely custom protocol, and are not limited to a set of coprocessor protocol 'primitives'. Instead, memory-mapped or I/O-mapped interfaces may use all applicable instructions for high-speed coprocessor communication. The **BUSY#** and **ERROR#** inputs of the 386 SX Microprocessor may also be used for the custom coprocessor interface, if such hardware assist is desired. These signals can be tested by the **WAIT** opcode (9BH). The **WAIT** instruction will wait until the **BUSY#** input is inactive (interruptable by an **NMI** or enabled **INTR** input), but generates an exception 16 fault if the **ERROR#** pin is active when the **BUSY#** goes (or is) inactive. If the custom coprocessor interface is memory-mapped, protection of the addresses used for the interface can be provided with the 386 SX CPU's on-chip paging or segmentation mechanisms. If the custom interface is I/O-mapped, protection of the interface can be provided with the **IOPL** (I/O Privilege Level) mechanism.

The 387™ SX numeric coprocessor interface is I/O mapped as shown in Table 5.8. Note that the 387™ SX coprocessor interface addresses are beyond the 0H-0FFFFH range for programmed I/O. When the 386 SX Microprocessor supports the 387™ SX coprocessor, the 386 SX Microprocessor automatically generates bus cycles to the coprocessor interface addresses.

Table 5.8. Numeric Coprocessor Port Addresses

Address in 386™ SX CPU I/O Space	387™ SX Coprocessor Register
8000F8H	Opcode Register
8000FCH/8000FEH*	Operand Register

*Generated as 2nd bus cycle during Dword transfer.

To correctly map the 387™ SX registers to the appropriate I/O addresses, connect the **CMD0** and **CMD1** lines of the 387™ SX as listed in Table 5.9.

Table 5.9. Connections for CMD0 and CMD1 Inputs for the 387™ SX

Signal	Connection
CMD0	Connect directly to 386™ SX CPU A2 signal
CMD1	Connect to ground.

Software Testing for Coprocessor Presence

When software is used to test for coprocessor (387 SX) presence, it should use only the following coprocessor opcodes: FINIT, FNINIT, FSTCW mem, FSTSW mem and FSTSW AX. To use other coprocessor opcodes when a coprocessor is known to be not present, first set EM = 1 in the 386 SX CPU's CR0 register.

6.0 PACKAGE THERMAL SPECIFICATIONS

The 386 SX Microprocessor is specified for operation when case temperature is within the range of 0°C–85°C. The case temperature may be measured in any environment, to determine whether the 386 SX Microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature is guaranteed as long as T_c is not violated. The ambient temperature can be calculated from the θ_{jc} and θ_{ja} from the following equations:

$$T_j = T_c + P \cdot \theta_{jc}$$

$$T_a = T_j - P \cdot \theta_{ja}$$

$$T_c = T_a + P \cdot [\theta_{ja} - \theta_{jc}]$$

Values for θ_{ja} and θ_{jc} are given in table 6.1 for the 100 lead fine pitch. θ_{ja} is given at various airflows. Table 6.2 shows the maximum T_a allowable (without exceeding T_c) at various airflows. Note that T_a can be improved further by attaching 'fins' or a 'heat sink' to the package.

Table 6.1. Thermal Resistances (°C/Watt) θ_{jc} and θ_{ja} .

Package	θ_{jc}	θ_{ja} versus Airflow - ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100 Lead Fine Pitch	7	33	27	24	21	18	17

Table 6.2. Maximum T_a at various airflows.

Package	Frequency	T_A (°C) versus Airflow - ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100L PQFP Fine Pitch	16 MHz	49	58	62	66	70	71
	20 MHz	45	55	59	64	68	70

Max. T_A calculated at 5.0V and max I_{CC} .

7.0 ELECTRICAL SPECIFICATIONS

The following sections describe recommended electrical connections for the 386 SX Microprocessor, and its electrical specifications.

7.1 Power and Grounding

The 386 SX Microprocessor is implemented in CHMOS III technology and has modest power requirements. However, its high clock frequency and 47 output buffers (address, data, control, and HLDA) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 14 Vcc and 18 Vss pins separately feed functional units of the 386 SX Microprocessor.

Power and ground connections must be made to all external Vcc and Vss pins of the 386 SX Microprocessor. On the circuit board, all Vcc pins should be connected on a Vcc plane and all Vss pins should be connected on a GND plane.

POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitors should be placed near the 386 SX Microprocessor. The 386 SX Microprocessor driving its 24-bit address bus and 16-bit data bus at high frequencies can cause transient power surges, particularly when driving large capacitive loads. Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the 386 SX Microprocessor and decoupling capacitors as much as possible.

Table 7.1. Recommended Resistor Pull-ups to Vcc

Pin	Signal	Pull-up Value	Purpose
16	ADS#	20 K-Ohm ± 10%	Lightly pull ADS# inactive during 386™ SX CPU hold acknowledge states
26	LOCK#	20 K-Ohm ± 10%	Lightly pull LOCK# inactive during 386™ SX CPU hold acknowledge states

RESISTOR RECOMMENDATIONS

The ERROR# and BUSY# inputs have internal pull-up resistors of approximately 20 K-Ohms and the PEREQ input has an internal pull-down resistor of approximately 20 K-Ohms built into the 386 SX Microprocessor to keep these signals inactive when the 387 SX is not present in the system (or temporarily removed from its socket).

In typical designs, the external pull-up resistors shown in Table 7.1 are recommended. However, a particular design may have reason to adjust the resistor values recommended here, or alter the use of pull-up resistors in other ways.

OTHER CONNECTION RECOMMENDATIONS

For reliable operation, always connect unused inputs to an appropriate signal level. N/C pins should always remain **unconnected**. **Connection of N/C pins to Vcc or Vss will result in component malfunction or incompatibility with future steppings of the 386 SX Microprocessor.**

Particularly when not using interrupts or bus hold (as when first prototyping), prevent any chance of spurious activity by connecting these associated inputs to GND:

Pin	Signal
40	INTR
38	NMI
4	HOLD

If not using address pipelining, connect pin 6, NA#, through a pull-up in the range of 20 K-Ohms to Vcc.

7.2 Maximum Ratings

Table 7.2. Maximum Ratings

Parameter	Maximum Rating
Storage temperature	-65 °C to 150 °C
Case temperature under bias	-65 °C to 110 °C
Supply voltage with respect to Vss	-.5V to 6.5V
Voltage on other pins	-.5V to (Vcc + .5)V

Table 7.2 gives stress ratings only, and functional operation at the maximums is not guaranteed. Functional operating conditions are given in section 7.3, **D.C. Specifications**, and section 7.4, **A.C. Specifications**.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the 386 SX Microprocessor contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

7.3 D.C. Specifications

 Functional operating range: $V_{CC} = 5V \pm 10\%$; $T_{CASE} = 0^{\circ}C$ to $85^{\circ}C$
Table 7.3. D.C. Characteristics

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input LOW Voltage	-0.3	+0.8	V	
V_{IH}	Input HIGH Voltage	2.0	$V_{CC} + 0.3$	V	
V_{ILC}	CLK2 Input LOW Voltage	-0.3	+0.8	V	
V_{IHC}	CLK2 Input HIGH Voltage	$V_{CC} - 0.8$	$V_{CC} + 0.3$	V	
V_{OL}	Output LOW Voltage $I_{OL} = 4mA$: $I_{OL} = 5mA$:		0.45 0.45	V V	
	A ₂₃ -A ₁ , D ₁₅ -D ₀ BHE #, BLE #, W/R #, D/C #, M/IO #, LOCK #, ADS #, HLDA				
V_{OH}	Output high voltage $I_{OH} = -1mA$: $I_{OH} = -0.2mA$: $I_{OH} = -0.9mA$:	2.4 $V_{CC} - 0.5$ 2.4		V V V	
	A ₂₃ -A ₁ , D ₁₅ -D ₀ BHE #, BLE #, W/R #, D/C #, M/IO #, LOCK #, ADS #, HLDA				
	$I_{OH} = -0.18mA$: BHE #, BLE #, W/R #, D/C #, M/IO #, LOCK #, ADS #, HLDA	$V_{CC} - 0.5$			
I_{LI}	Input leakage current (for all pins except PEREQ, BUSY # and ERROR #)		± 15	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{IH}	Input Leakage Current (PEREQ pin)		200	μA	$V_{IH} = 2.4V$, Note 1
I_{IL}	Input Leakage Current (BUSY # and ERROR # pins)		-400	μA	$V_{IL} = 0.45V$, Note 2
I_{LO}	Output leakage current		± 15	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
I_{CC}	Supply Current CLK2 = 32 MHz CLK2 = 40 MHz		275 305	mA mA	I_{CC} typ = 175 mA, Note 3 I_{CC} typ = 200 mA, Note 3
C_{IN}	Input capacitance		10	pF	$F_c = 1$ MHz, Note 4
C_{OUT}	Output or I/O capacitance		12	pF	$F_c = 1$ MHz, Note 4
C_{CLK}	CLK2 Capacitance		20	pF	$F_c = 1$ MHz, Note 4

Tested at the minimum operating frequency of the part.

NOTES:

- PEREQ input has an internal pull-down resistor.
- BUSY # and ERROR # inputs each have an internal pull-up resistor.
- I_{CC} max measurement at worst case load, frequency, V_{CC} and temperature.
- Not 100% tested.

7.4 A.C. Specifications

The A.C. specifications given in Table 7.4 consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the CLK2 rising edge crossing the 2.0V level.

A.C. spec measurement is defined by Figure 7.1. Inputs must be driven to the voltage levels indicated by Figure 7.1 when A.C. specifications are measured. Output delays are specified with minimum and maximum limits measured as shown. The minimum delay times are hold times provided to external circuitry. Input setup and hold times are specified

as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct operation.

Outputs NA#, W/R#, D/C#, M/IO#, LOCK#, BHE#, BLE#, A₂₃-A₁ and HLDA only change at the beginning of phase one. D₁₅-D₀ (write cycles) only change at the beginning of phase two. The READY#, HOLD, BUSY#, ERROR#, PEREQ and D₁₅-D₀ (read cycles) inputs are sampled at the beginning of phase one. The NA#, INTR and NMI inputs are sampled at the beginning of phase two.

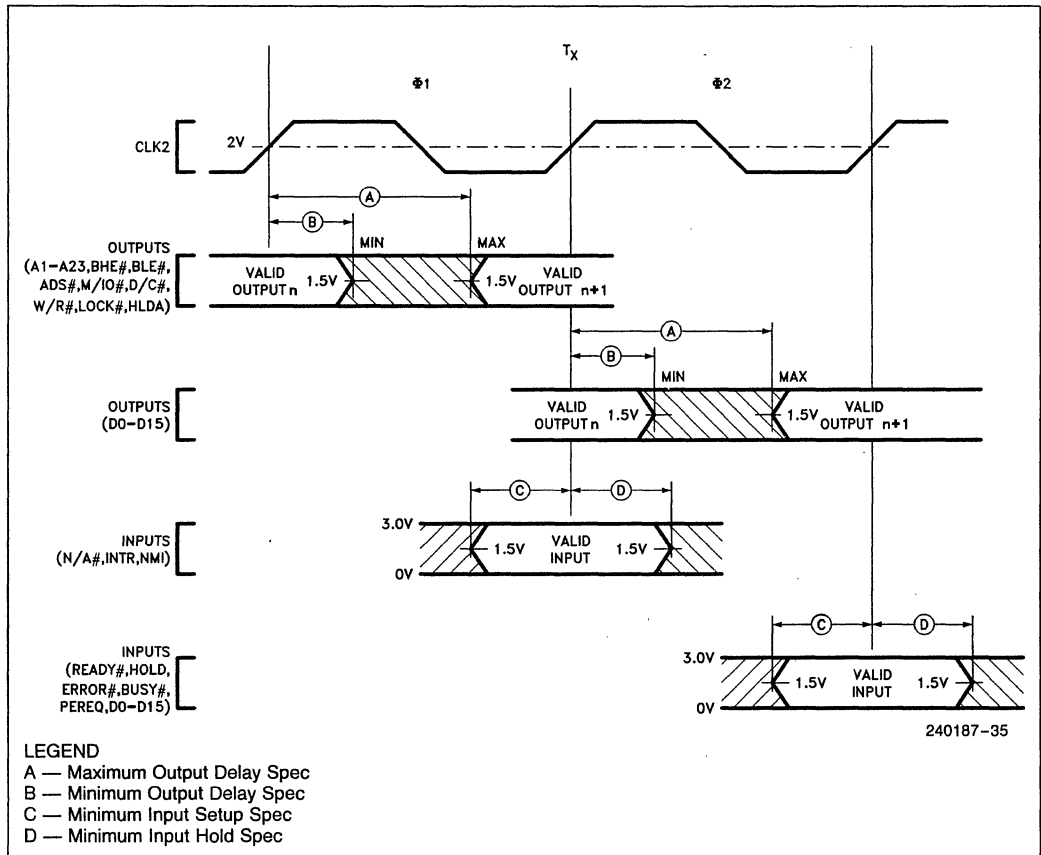


Figure 7.1. Drive Levels and Measurement Points for A.C. Specifications

A.C. SPECIFICATONS TABLES

 Functional operating range: $V_{CC} = 5V \pm 10\%$; $T_{CASE} = 0^{\circ}C$ to $85^{\circ}C$
Table 7.4. A.C. Characteristics at 16 MHz

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Operating frequency	4	16	MHz		Half CLK2 Freq
t_1	CLK2 period	31	125	ns	7.3	
t_{2a}	CLK2 HIGH time	9		ns	7.3	at 2V ⁽³⁾
t_{2b}	CLK2 HIGH time	5		ns	7.3	at $(V_{CC} - 0.8)V$ ⁽³⁾
t_{3a}	CLK2 LOW time	9		ns	7.3	at 2V ⁽³⁾
t_{3b}	CLK2 LOW time	7		ns	7.3	at 0.8V ⁽³⁾
t_4	CLK2 fall time		8	ns	7.3	$(V_{CC} - 0.8)V$ to 0.8V ⁽³⁾
t_5	CLK2 rise time		8	ns	7.3	0.8V to $(V_{CC} - 0.8)V$ ⁽³⁾
t_6	A ₂₃ -A ₁ valid delay	4	36	ns	7.5	$C_L = 120pF$ ⁽⁴⁾
t_7	A ₂₃ -A ₁ float delay	4	40	ns	7.6	(Note 1)
t_8	BHE #, BLE #, LOCK # valid delay	4	36	ns	7.5	$C_L = 75pF$ ⁽⁴⁾
t_9	BHE #, BLE #, LOCK # float delay	4	40	ns	7.6	(Note 1)
t_{10}	W/R #, M/IO #, D/C #, ADS # valid delay	6	33	ns	7.5	$C_L = 75pF$ ⁽⁴⁾
t_{11}	W/R #, M/IO #, D/C #, ADS # float delay	6	35	ns	7.6	(Note 1)
t_{12}	D ₁₅ -D ₀ Write Data Valid Delay	4	40	ns	7.5	$C_L = 120pF$ ⁽⁴⁾
t_{13}	D ₁₅ -D ₀ Write Data Float Delay	4	35	ns	7.6	(Note 1)
t_{14}	HLDA valid delay	6	33	ns	7.5	$C_L = 75pF$ ⁽⁴⁾
t_{15}	NA # setup time	5		ns	7.4	
t_{16}	NA # hold time	21		ns	7.4	
t_{19}	READY # setup time	19		ns	7.4	
t_{20}	READY # hold time	4		ns	7.4	
t_{21}	D ₁₅ -D ₀ Read Data setup time	9		ns	7.4	
t_{22}	D ₁₅ -D ₀ Read Data hold time	6		ns	7.4	
t_{23}	HOLD setup time	26		ns	7.4	
t_{24}	HOLD hold time	5		ns	7.4	
t_{25}	RESET setup time	13		ns	7.7	
t_{26}	RESET hold time	4		ns	7.7	

Functional operating range: $V_{CC} = 5V \pm 10\%$; $T_{CASE} = 0^{\circ}C$ to $85^{\circ}C$
Table 7.4. A.C. Characteristics at 16 MHz (Continued)

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t ₂₇	NMI, INTR setup time	16		ns	7.4	(Note 2)
t ₂₈	NMI, INTR hold time	16		ns	7.4	(Note 2)
t ₂₉	PEREQ, ERROR #, BUSY # setup time	16		ns	7.4	(Note 2)
t ₃₀	PEREQ, ERROR #, BUSY # hold time	5		ns	7.4	(Note 2)

Table 7.5. A.C. Characteristics at 20 MHz

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Operating Frequency	4	20	MHz		Half CLK2 Frequency
t ₁	CLK2 period	25	125	ns	7.3	
t _{2a}	CLK2 HIGH time	8		ns	7.3	at 2V (Note 3)
t _{2b}	CLK2 HIGH time	5		ns	7.3	at ($V_{CC} - 0.8V$) (Note 3)
t _{3a}	CLK2 LOW time	8		ns	7.3	at 2V (Note 3)
t _{3b}	CLK2 LOW time	6		ns	7.3	at 0.8V (Note 3)
t ₄	CLK2 fall time		8	ns	7.3	($V_{CC} - 0.8V$) to 0.8V (Note 3)
t ₅	CLK2 rise time		8	ns	7.3	0.8V to ($V_{CC} - 0.8V$) (Note 3)
t ₆	A23-A1 valid delay	4	30	ns	7.5	$C_L = 120$ pF (Note 4)
t ₇	A23-A1 float delay	4	32	ns	7.6	(Note 1)
t ₈	BHE #, BLE #, LOCK # valid delay	4	30	ns	7.5	$C_L = 75$ pF (Note 4)
t ₉	BHE #, BLE #, LOCK # float delay	4	32	ns	7.6	(Note 1)
t _{10a}	M/IO #, D/C # valid delay	6	28	ns	7.5	$C_L = 75$ pF (Note 4)
t _{10b}	W/R #, ADS # valid delay	6	26	ns	7.5	$C_L = 75$ pF (Note 4)
t ₁₁	W/R #, M/IO #, D/C #, ADS # float delay	6	30	ns	7.6	(Note 1)
t ₁₂	D15-D0 Write Data Valid Delay	4	38	ns	7.5	$C_L = 120$ pF
t ₁₃	D15-D0 Write Data Float Delay	4	27	ns	7.6	(Note 1)
t ₁₄	HLDA valid delay	4	28	ns	7.5	$C_L = 75$ pF (Note 4)
t ₁₅	NA # setup time	5		ns	7.4	
t ₁₆	NA # hold time	12		ns	7.4	

Table 7.5. A.C. Characteristics at 20 MHz (Continued)

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t19	READY # setup time	12		ns	7.4	
t20	READY # hold time	4		ns	7.4	
t21	D15–D0 Read Data setup time	9		ns	7.4	
t22	D15–D0 Read Data hold time	6		ns	7.4	
t23	HOLD setup time	17		ns	7.4	
t24	HOLD hold time	5		ns	7.4	
t25	RESET setup time	12		ns	7.7	
t26	RESET hold time*	4		ns	7.7	
t27	NMI, INTR setup time	16		ns	7.4	(Note 2)
t28	NMI, INTR hold time	16		ns	7.4	(Note 2)
t29	PEREQ, ERROR#, BUSY # setup time	14		ns	7.4	(Note 2)
t30	PEREQ, ERROR#, BUSY # hold time	5		ns	7.4	(Note 2)

NOTES:

1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.
3. These are not tested. They are guaranteed by design characterization.
4. Tested with C_L set at 50 pf and derated to support the indicated distributed capacitive load. See Figures 7.8 though 7.10 for the capacitive derating curve.

A.C. TEST LOADS

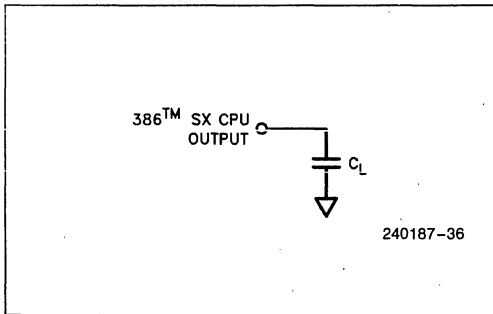


Figure 7.2. A.C. Test Loads

A.C. TIMING WAVEFORMS

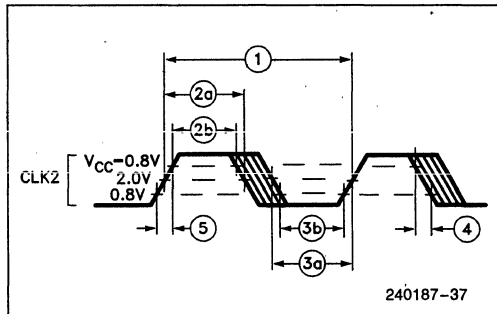


Figure 7.3. CLK2 Waveform

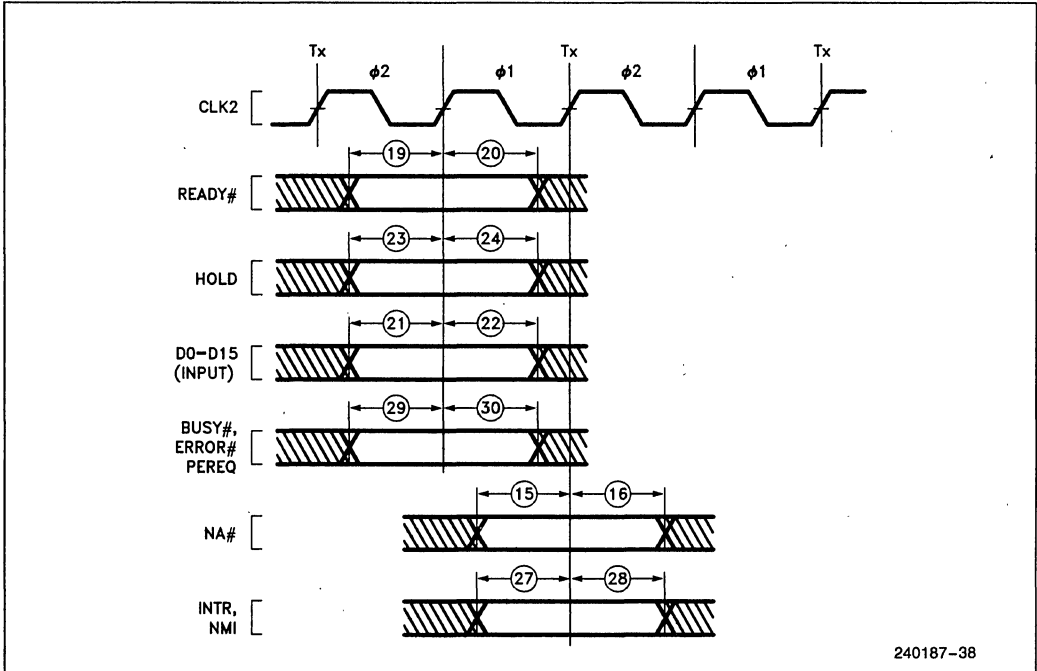


Figure 7.4. A.C. Timing Waveforms—Input Setup and Hold Timing

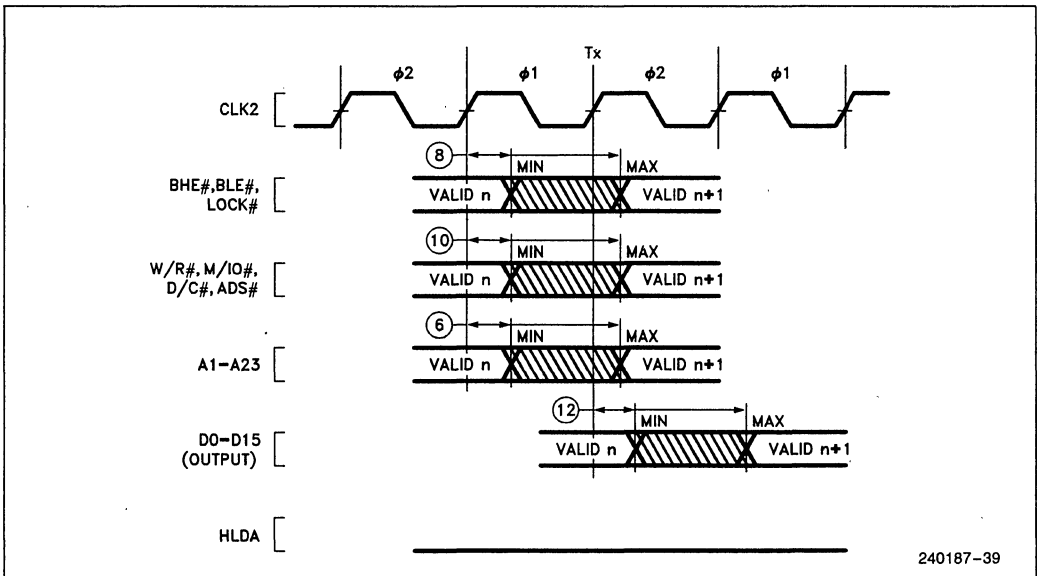


Figure 7.5. A.C. Timing Waveforms—Output Valid Delay Timing

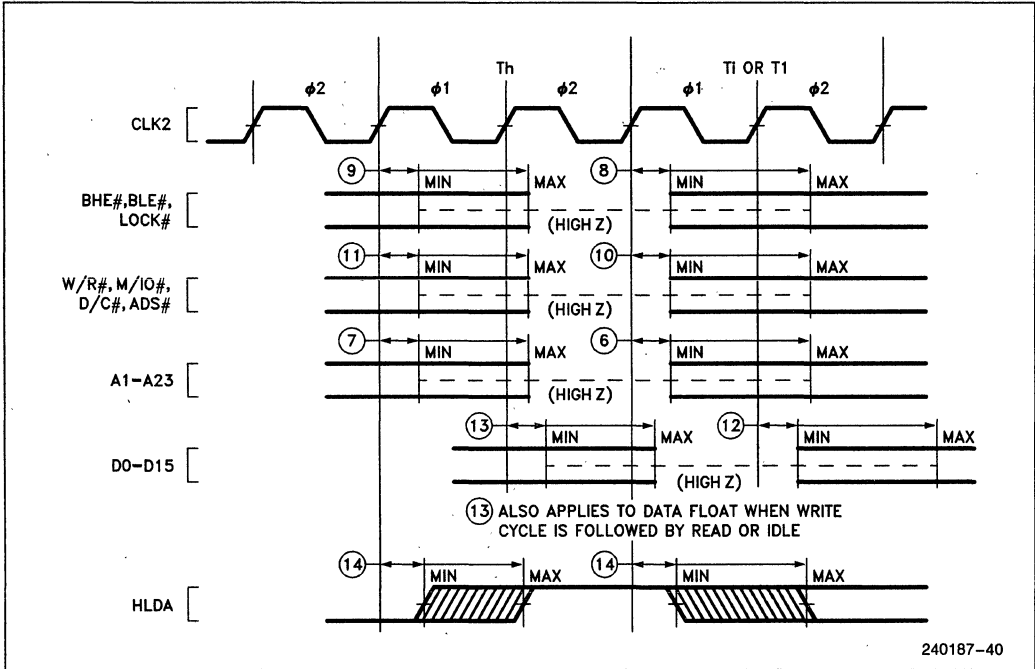


Figure 7.6. A.C. Timing Waveforms—Output Float Delay and HLDA Valid Delay Timing

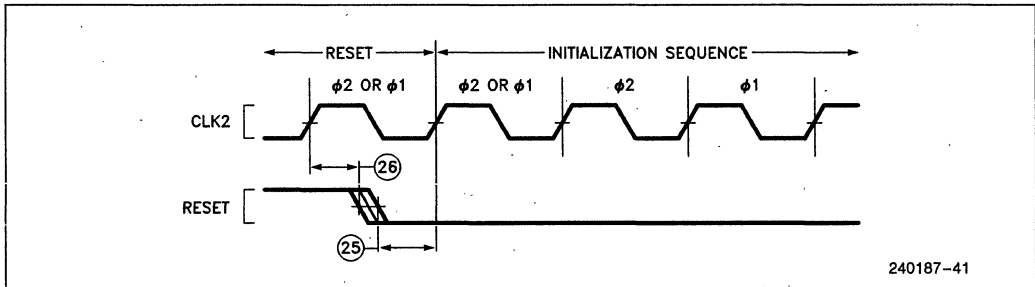


Figure 7.7. A.C. Timing Waveforms—RESET Setup and Hold Timing and Internal Phase

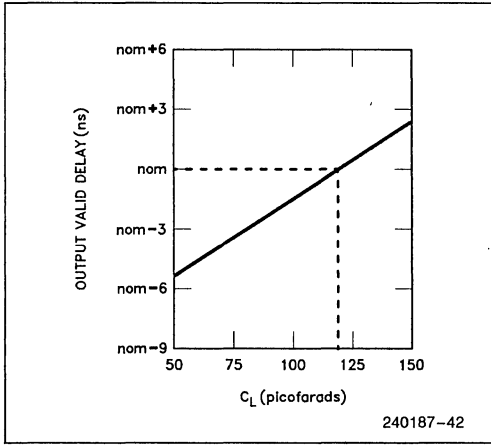


Figure 7.8. Typical Output Valid Delay versus Load Capacitance at Maximum Operating Temperature ($C_L = 120$ pF)

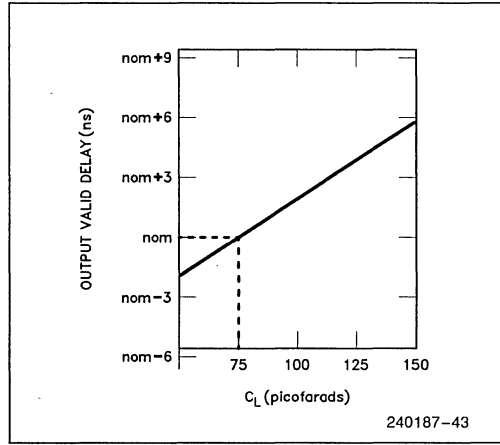


Figure 7.9. Typical Output Valid Delay versus Load Capacitance at Maximum Operating Temperature ($C_L = 75$ pF)

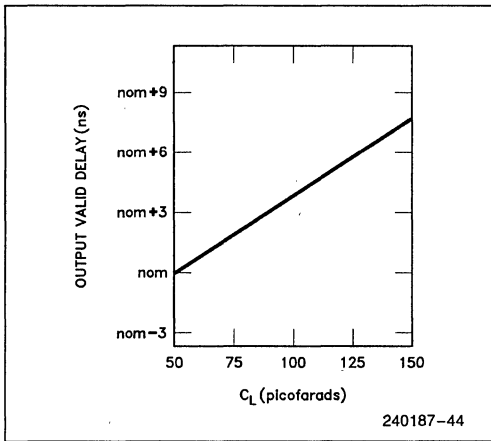


Figure 7.10. Typical Output Valid Delay versus Load Capacitance at Maximum Operating Temperature ($C_L = 50$ pF)

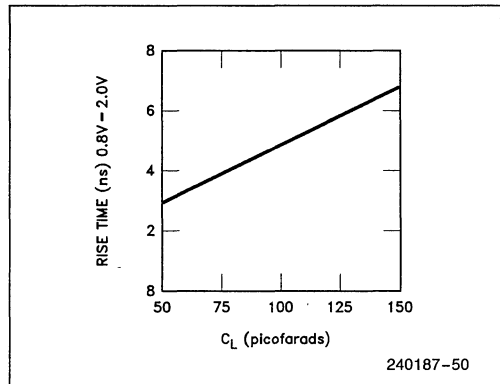


Figure 7.11. Typical Output Rise Time versus Load Capacitance at Maximum Operating Temperature

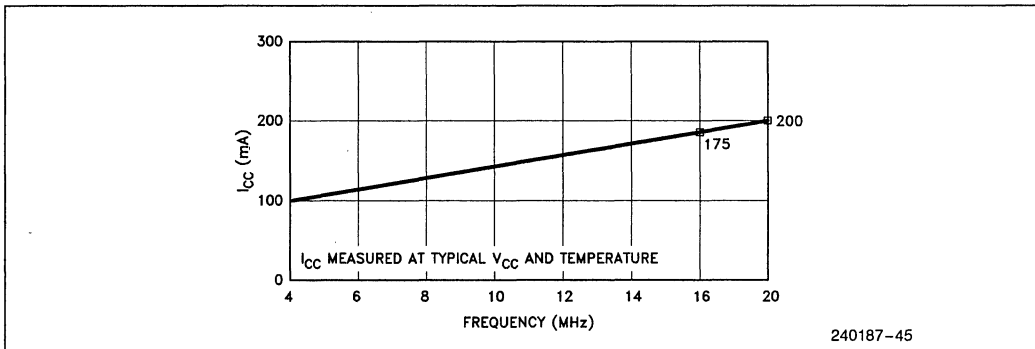


Figure 7.12. Typical Icc vs Frequency

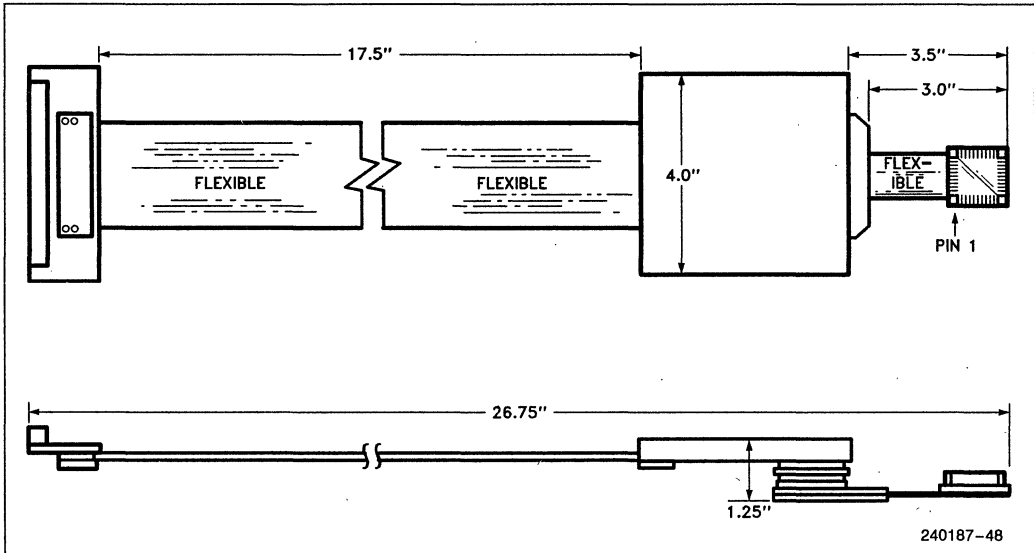


Figure 7.13. Preliminary ICETM-386 SX Emulator User Cable with PQFP Adapter

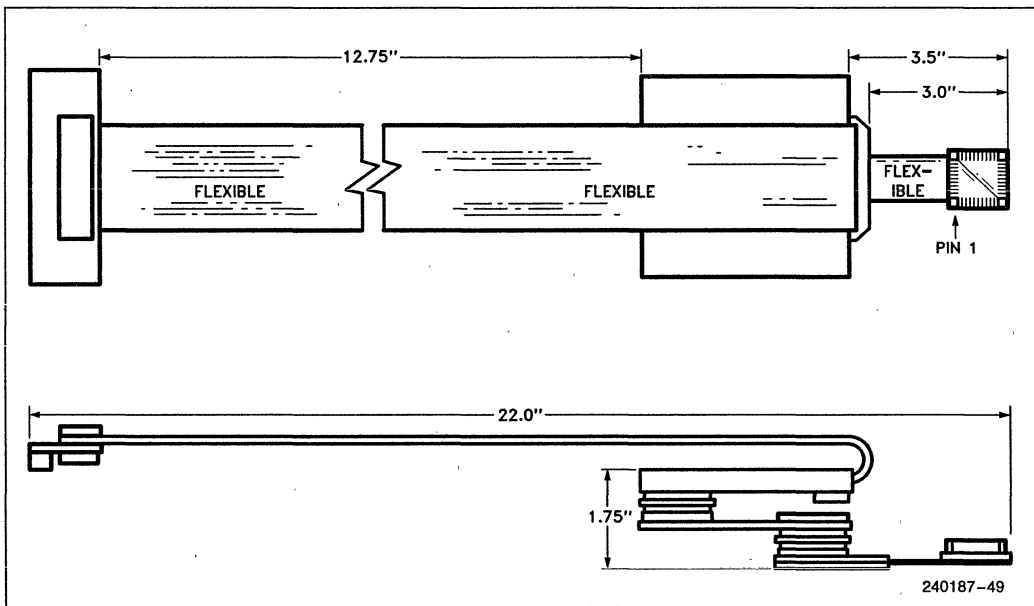


Figure 7.14. Preliminary ICETM-386 SX Emulator User Cable with OIB and PQFP Adapter

7.5 Designing for ICETM-386 SX Emulator (Advanced Data)

The 386 SX CPU's in-circuit emulator product is the ICETM-386 SX emulator. Use of the emulator requires the target system to provide a socket that is compatible with the ICE-386 SX emulator. The ICE-386 SX offers a 100-pin fine pitch flat-pack probe for emulating user systems. The 100-pin fine pitch flat-pack probe requires a socket, called the 100-pin PQFP, which is available from 3M text-tool (part number 2-0100-07243-000). The ICE-386 SX emulator probe attaches to the target system via an adapter which replaces the 386 SX CPU component in the target system. Because of the high operating frequency of 386 SX CPU systems and of the ICE-386 SX emulator, there is no buffering between the 386 SX CPU emulation processor in the ICE-386 SX emulator probe and the target system. A direct result of the non-buffered interconnect is that the ICE-386 SX emulator shares the address and data bus with the user's system, and the RESET signal is intercepted by the ICE emulator hardware. In order for the ICE-386 SX emulator to be functional in the user's system without the Optional Isolation Board (OIB) the designer must be aware of the following conditions:

1. The bus controller must only enable data transceivers onto the data bus during valid read cycles of the 386 SX CPU, other local devices or other bus masters.
2. Before another bus master drives the local processor address bus, the other master must gain control of the address bus by asserting HOLD and receiving the HLDA response.
3. The emulation processor receives the RESET signal 2 or 4 CLK2 cycles later than an 386 SX CPU would, and responds to RESET later. Correct phase of the response is guaranteed.

In addition to the above considerations, the ICE-386 SX emulator processor module has several electrical and mechanical characteristics that should be taken into consideration when designing the 386 SX CPU system.

Capacitive Loading: ICE-386 SX adds up to 27 pF to each 386 SX CPU signal.

Drive Requirements: ICE-386 SX adds one FAST TTL load on the CLK2, control, address, and data lines. These loads are within the processor module and are driven by the 386 SX CPU emulation processor, which has standard drive and loading capability listed in Tables 7.3 and 7.4.

Power Requirements: For noise immunity and CMOS latch-up protection the ICE-386 SX emulator processor module is powered by the user system.

The circuitry on the processor module draws up to 1.4A including the maximum 386 SX CPU I_{CC} from the user 386 SX CPU socket.

386 SX CPU Location and Orientation: The ICE-386 SX emulator processor module may require lateral clearance. Figure 7.12 shows the clearance requirements of the iMP adapter. The optional isolation board (OIB), which provides extra electrical buffering and has the same lateral clearance requirements as Figure 7.13, adds an additional 0.5 inches to the vertical clearance requirement. This is illustrated in Figure 7.14.

Optional Isolation Board (OIB) and the CLK2 speed reduction: Due to the unbuffered probe design, the ICE-386 SX emulator is susceptible to errors on the user's bus. The OIB allows the ICE-386 SX emulator to function in user systems with faults (shorted signals, etc.). After electrical verification the OIB may be removed. When the OIB is installed, the user system must have a maximum CLK2 frequency of 20 MHz.

8.0 DIFFERENCES BETWEEN THE 386 SX CPU AND THE 386 DX CPU

The following are the major differences between the 386 SX CPU and the 386 DX CPU:

1. The 386 SX CPU generates byte selects on BHE# and BLE# (like the 8086 and 80286) to distinguish the upper and lower bytes on its 16-bit data bus. The 386 DX CPU uses four byte selects, BE0#-BE3#, to distinguish between the different bytes on its 32-bit bus.
2. The 386 SX CPU has no bus sizing option. The 386 DX CPU can select between either a 32-bit bus or a 16-bit bus by use of the BS16# input. The 386 SX CPU has a 16-bit bus size.
3. The NA# pin operation in the 386 SX CPU is identical to that of the NA# pin on the 386 DX CPU with one exception: the 386 DX CPU NA# pin cannot be activated on 16-bit bus cycles (where BS16# is LOW in the 386 DX CPU case), whereas NA# can be activated on any 386 SX CPU bus cycle.
4. The contents of all 386 SX CPU registers at reset are identical to the contents of the 386 DX CPU registers at reset, except the DX register. The DX register contains a component-stepping identifier at reset, i.e.

in 386 DX CPU, DH = 3 indicates 386 DX CPU after reset

DL = revision number;

in 386 SX CPU, DH = 23H indicates 386 SX CPU after reset

DL = revision number.

5. The 386 DX CPU uses A_{31} and $M/IO\#$ as selects for the numerics coprocessor. The 386 SX CPU uses A_{23} and $M/IO\#$ as selects.
6. The 386 DX CPU prefetch unit fetches code in four-byte units. The 386 SX CPU prefetch unit reads two bytes as one unit (like the 80286). In BS16 mode, the 386 DX CPU takes two consecutive bus cycles to complete a prefetch request. If there is a data read or write request after the prefetch starts, the 386 DX CPU will fetch all four bytes before addressing the new request.
7. Both 386 DX CPU and 386 SX CPU have the same logical address space. The only difference is that the 386 DX CPU has a 32-bit physical address space and the 386 SX CPU has a 24-bit physical address space. The 386 SX CPU has a physical memory address space of up to 16 megabytes instead of the 4 gigabytes available to the 386 DX CPU. Therefore, in 386 SX CPU systems, the operating system must be aware of this physical memory limit and should allocate memory for applications programs within this limit. If a 386 DX CPU system uses only the lower 16 megabytes of physical address, then there will be no extra effort required to migrate 386 DX CPU software to the 386 SX CPU. Any application which uses more than 16 megabytes of memory can run on the 386 SX CPU if the operating system utilizes the 386 SX CPU's paging mechanism. In spite of this difference in physical address space, the 386 SX CPU and 386 DX CPU can run the same operating systems and applications within their respective physical memory constraints.

9.0 INSTRUCTION SET

This section describes the instruction set. Table 9.1 lists all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in the following sections, which completely describe the encoding structure and the definition of all fields occurring within instructions.

9.1 386 SX CPU Instruction Encoding and Clock Count Summary

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Table 9.1 be-

low, by the processor clock period (e.g. 62.5 ns for an 386 SX Microprocessor operating at 16 MHz). The actual clock count of an 386 SX Microprocessor program will average 5% more than the calculated clock count due to instruction sequences which execute faster than they can be fetched from memory.

Instruction Clock Count Assumptions

1. The instruction has been prefetched, decoded, and is ready for execution.
2. Bus cycles do not require wait states.
3. There are no local bus HOLD requests delaying processor access to the bus.
4. No exceptions are detected during instruction execution.
5. If an effective address is calculated, it does not use two general register components. One register, scaling and displacement can be used within the clock counts shown. However, if the effective address calculation uses two general register components, add 1 clock to the clock count shown.

Instruction Clock Count Notation

1. If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand.
2. n = number of times repeated.
3. m = number of components in the next instruction executed, where the entire displacement (if any) counts as one component, the entire immediate data (if any) counts as one component, and all other bytes of the instruction and prefix(es) each count as one component.

Misaligned or 32-Bit Operand Accesses

- If instructions accesses a misaligned 16-bit operand or 32-bit operand on even address add:
 - 2* clocks for read or write
 - 4** clocks for read and write
- If instructions accesses a 32-bit operand on odd address add:
 - 4* clocks for read or write
 - 8** clocks for read and write

Wait States

Wait states add 1 clock per wait state to instruction execution for each data access.

Table 9-1. Instruction Set Clock Count Summary

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
GENERAL DATA TRANSFER					
MOV = Move:					
Register to Register/Memory	1 0 0 0 1 0 0 w mod reg r/m	2/2	2/2*	b	h
Register/Memory to Register	1 0 0 0 1 0 1 w mod reg r/m	2/4	2/4*	b	h
Immediate to Register/Memory	1 1 0 0 0 1 1 w mod 0 0 0 r/m immediate data	2/2	2/2*	b	h
Immediate to Register (short form)	1 0 1 1 w reg immediate data	2	2		
Memory to Accumulator (short form)	1 0 1 0 0 0 0 w full displacement	4*	4*	b	h
Accumulator to Memory (short form)	1 0 1 0 0 0 1 w full displacement	2*	2*	b	h
Register Memory to Segment Register	1 0 0 0 1 1 1 0 mod sreg3 r/m	2/5	22/23	b	h, i, j
Segment Register to Register/Memory	1 0 0 0 1 1 0 0 mod sreg3 r/m	2/2	2/2	b	h
MOVX = Move With Sign Extension					
Register From Register/Memory	0 0 0 0 1 1 1 1 1 0 1 1 1 1 1 w mod reg r/m	3/6*	3/6*	b	h
MOVZX = Move With Zero Extension					
Register From Register/Memory	0 0 0 0 1 1 1 1 1 0 1 1 0 1 1 w mod reg r/m	3/6*	3/6*	b	h
PUSH = Push:					
Register/Memory	1 1 1 1 1 1 1 1 mod 1 1 0 r/m	5/7*	7/9*	b	h
Register (short form)	0 1 0 1 0 reg	2	4	b	h
Segment Register (ES, CS, SS or DS) (short form)	0 0 0 sreg2 1 1 0	2	4	b	h
Segment Register (ES, CS, SS, DS, FS or GS)	0 0 0 0 1 1 1 1 1 0 sreg3 0 0 0	2	4	b	h
Immediate	0 1 1 0 1 0 s 0 immediate data	2	4	b	h
PUSHA = Push All					
	0 1 1 0 0 0 0 0	18	34	b	h
POP = Pop					
Register/Memory	1 0 0 0 1 1 1 1 mod 0 0 0 r/m	5/7	7/9	b	h
Register (short form)	0 1 0 1 1 reg	6	6	b	h
Segment Register (ES, CS, SS or DS) (short form)	0 0 0 sreg2 1 1 1	7	25	b	h, i, j
Segment Register (ES, CS, SS or DS), FS or GS	0 0 0 0 1 1 1 1 1 0 sreg3 0 0 1	7	25	b	h, i, j
POPA = Pop All					
	0 1 1 0 0 0 0 1	24	40	b	h
XCHG = Exchange					
Register/Memory With Register	1 0 0 0 0 1 1 w mod reg r/m	3/5**	3/5**	b, f	f, h
Register With Accumulator (short form)	1 0 0 1 0 reg	3	3		
IN = Input from:					
Fixed Port	1 1 1 0 0 1 0 w port number	†26	12*	6*/26*	s/t, m
Variable Port	1 1 1 0 1 1 0 w	†27	13*	7*/27*	s/t, m
OUT = Output to:					
Fixed Port	1 1 1 0 0 1 1 w port number	†24	10*	4*/24*	s/t, m
Variable Port	1 1 1 0 1 1 1 w	†25	11*	5*/25*	s/t, m
LEA = Load EA to Register					
	1 0 0 0 1 1 0 1 mod reg r/m	2	2		

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
SEGMENT CONTROL					
LDS = Load Pointer to DS	11000101 mod reg r/m	7*	26*/28*	b	h, i, j
LES = Load Pointer to ES	11000100 mod reg r/m	7*	26*/28*	b	h, i, j
LFS = Load Pointer to FS	00001111 10110100 mod reg r/m	7*	29*/31*	b	h, i, j
LGS = Load Pointer to GS	00001111 10110101 mod reg r/m	7*	26*/28*	b	h, i, j
LSS = Load Pointer to SS	00001111 10110010 mod reg r/m	7*	26*/28*	b	h, i, j
FLAG CONTROL					
CLC = Clear Carry Flag	11111000	2			
CLD = Clear Direction Flag	11111100	2			
CLI = Clear Interrupt Enable Flag	11111010	8	8		m
CLTS = Clear Task Switched Flag	00001111 00000110	5	5	c	l
CMC = Complement Carry Flag	11110101	2	2		
LAHF = Load AH Into Flag	10011111	2	2		
POPF = Pop Flags	10011101	5	5	b	h, n
PUSHF = Push Flags	10011100	4	4	b	h
SAHF = Store AH Into Flags	10011110	3	3		
STC = Set Carry Flag	11111001	2	2		
STD = Set Direction Flag	11111101				
STI = Set Interrupt Enable Flag	11111011	8	8		m
ARITHMETIC					
ADD = Add					
Register to Register	000000dw mod reg r/m	2	2		
Register to Memory	0000000w mod reg r/m	7**	7**	b	h
Memory to Register	0000001w mod reg r/m	6*	6*	b	h
Immediate to Register/Memory	100000sw mod 000 r/m immediate data	2/7**	2/7**	b	h
Immediate to Accumulator (short form)	0000010w immediate data	2	2		
ADC = Add With Carry					
Register to Register	000100dw mod reg r/m	2	2		
Register to Memory	0001000w mod reg r/m	7**	7**	b	h
Memory to Register	0001001w mod reg r/m	6*	6*	b	h
Immediate to Register/Memory	100000sw mod 010 r/m immediate data	2/7**	2/7**	b	h
Immediate to Accumulator (short form)	0001010w immediate data	2	2		
INC = Increment					
Register/Memory	1111111w mod 000 r/m	2/6**	2/6**	b	h
Register (short form)	01000 reg	2	2		
SUB = Subtract					
Register from Register	001010dw mod reg r/m	2	2		

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
ARITHMETIC (Continued)					
Register from Memory	0010100w mod reg r/m	7**	7**	b	h
Memory from Register	0010101w mod reg r/m	6*	6*	b	h
Immediate from Register/Memory	100000s w mod 101 r/m	2/7**	2/7**	b	h
Immediate from Accumulator (short form)	0010110w immediate data	2	2		
SBB = Subtract with Borrow					
Register from Register	000110dw mod reg r/m	2	2		
Register from Memory	0001100w mod reg r/m	7**	7**	b	h
Memory from Register	0001101w mod reg r/m	6*	6*	b	h
Immediate from Register/Memory	100000s w mod 011 r/m	2/7**	2/7**	b	h
Immediate from Accumulator (short form)	0001110w immediate data	2	2		
DEC = Decrement					
Register/Memory	1111111w reg 001 r/m	2/6	2/6	b	h
Register (short form)	01001 reg	2	2		
CMP = Compare					
Register with Register	001110dw mod reg r/m	2	2		
Memory with Register	0011100w mod reg r/m	5*	5*	b	h
Register with Memory	0011101w mod reg r/m	6*	6*	b	h
Immediate with Register/Memory	100000s w mod 111 r/m	2/5*	2/5*	b	h
Immediate with Accumulator (short form)	0011110w immediate data	2	2		
NEG = Change Sign					
	1111011w mod 011 r/m	2/6*	2/6*	b	h
AAA = ASCII Adjust for Add					
	00110111	4	4		
AAS = ASCII Adjust for Subtract					
	00111111	4	4		
DAA = Decimal Adjust for Add					
	00100111	4	4		
DAS = Decimal Adjust for Subtract					
	00101111	4	4		
MUL = Multiply (unsigned)					
Accumulator with Register/Memory	1111011w mod 100 r/m				
Multiplier-Byte		12-17/15-20*	12-17/15-20*	b, d	d, h
-Word		12-25/15-28*	12-25/15-28*	b, d	d, h
-Doubleword		12-41/17-46*	12-41/17-46*	b, d	d, h
IMUL = Integer Multiply (signed)					
Accumulator with Register/Memory	1111011w mod 101 r/m				
Multiplier-Byte		12-17/15-20*	12-17/15-20*	b, d	d, h
-Word		12-25/15-28*	12-25/15-28*	b, d	d, h
-Doubleword		12-41/17-46*	12-41/17-46*	b, d	d, h
Register with Register/Memory	00001111 10101111 mod reg r/m				
Multiplier-Byte		12-17/15-20*	12-17/15-20*	b, d	d, h
-Word		12-25/15-28*	12-25/15-28*	b, d	d, h
-Doubleword		12-41/17-46*	12-41/17-46*	b, d	d, h
Register/Memory with Immediate to Register	011010s 1 mod reg r/m				
-Word		13-26	13-26/14-27	b, d	d, h
-Doubleword		13-42	13-42/16-45	b, d	d, h

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES																	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode																
ARITHMETIC (Continued)																					
DIV = Divide (Unsigned)																					
Accumulator by Register/Memory	1111011w mod 110 r/m																				
Divisor—Byte		14/17	14/17	b,e	e,h																
—Word		22/25	22/25	b,e	e,h																
—Doubleword		38/43	38/43	b,e	e,h																
IDIV = Integer Divide (Signed)																					
Accumulator By Register/Memory	1111011w mod 111 r/m																				
Divisor—Byte		19/22	19/22	b,e	e,h																
—Word		27/30	27/30	b,e	e,h																
—Doubleword		43/48	43/48	b,e	e,h																
AAD = ASCII Adjust for Divide	11010101 00001010	19	19																		
AAM = ASCII Adjust for Multiply	11010100 00001010	17	17																		
CBW = Convert Byte to Word	10011000	3	3																		
CWD = Convert Word to Double Word	10011001	2	2																		
LOGIC																					
Shift Rotate Instructions																					
Not Through Carry (ROL, ROR, SAL, SAR, SHL, and SHR)																					
Register/Memory by 1	1101000w mod TTT r/m	3/7**	3/7**	b	h																
Register/Memory by CL	1101001w mod TTT r/m	3/7*	3/7*	b	h																
Register/Memory by Immediate Count	1100000w mod TTT r/m	3/7*	3/7*	b	h																
Through Carry (RCL and RCR)																					
Register/Memory by 1	1101000w mod TTT r/m	9/10*	9/10*	b	h																
Register/Memory by CL	1101001w mod TTT r/m	9/10*	9/10*	b	h																
Register/Memory by Immediate Count	1100000w mod TTT r/m	9/10*	9/10*	b	h																
<table border="0" style="width: 100%;"> <tr> <td style="text-align: center;">TTT</td> <td style="text-align: left;">Instruction</td> </tr> <tr> <td style="text-align: center;">000</td> <td>ROL</td> </tr> <tr> <td style="text-align: center;">001</td> <td>ROR</td> </tr> <tr> <td style="text-align: center;">010</td> <td>RCL</td> </tr> <tr> <td style="text-align: center;">011</td> <td>RCR</td> </tr> <tr> <td style="text-align: center;">100</td> <td>SHL/SAL</td> </tr> <tr> <td style="text-align: center;">101</td> <td>SHR</td> </tr> <tr> <td style="text-align: center;">111</td> <td>SAR</td> </tr> </table>						TTT	Instruction	000	ROL	001	ROR	010	RCL	011	RCR	100	SHL/SAL	101	SHR	111	SAR
TTT	Instruction																				
000	ROL																				
001	ROR																				
010	RCL																				
011	RCR																				
100	SHL/SAL																				
101	SHR																				
111	SAR																				
SHLD = Shift Left Double																					
Register/Memory by Immediate	00001111 10100100 mod reg r/m	3/7**	3/7**																		
Register/Memory by CL	00001111 10100101 mod reg r/m	3/7**	3/7**																		
SHRD = Shift Right Double																					
Register/Memory by Immediate	00001111 10101100 mod reg r/m	3/7**	3/7**																		
Register/Memory by CL	00001111 10101101 mod reg r/m	3/7**	3/7**																		
AND = And																					
Register to Register	001000d w mod reg r/m	2	2																		

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
LOGIC (Continued)					
Register to Memory	0010000w mod reg r/m	7**	7**	b	h
Memory to Register	0010001w mod reg r/m	6*	6*	b	h
Immediate to Register/Memory	1000000w mod 100 r/m immediate data	2/7*	2/7**	b	h
Immediate to Accumulator (Short Form)	0010010w immediate data	2	2		
TEST = And Function to Flags, No Result					
Register/Memory and Register	1000010w mod reg r/m	2/5*	2/5*	b	h
Immediate Data and Register/Memory	1111011w mod 000 r/m immediate data	2/5*	2/5*	b	h
Immediate Data and Accumulator (Short Form)	1010100w immediate data	2	2		
OR = Or					
Register to Register	000010dw mod reg r/m	2	2		
Register to Memory	0000100w mod reg r/m	7**	7**	b	h
Memory to Register	0000101w mod reg r/m	6*	6*	b	h
Immediate to Register/Memory	1000000w mod 001 r/m immediate data	2/7**	2/7**	b	h
Immediate to Accumulator (Short Form)	0000110w immediate data	2	2		
XOR = Exclusive Or					
Register to Register	001100dw mod reg r/m	2	2		
Register to Memory	0011000w mod reg r/m	7**	7**	b	h
Memory to Register	0011001w mod reg r/m	6*	6*	b	h
Immediate to Register/Memory	1000000w mod 110 r/m immediate data	2/7**	2/7**	b	h
Immediate to Accumulator (Short Form)	0011010w immediate data	2	2		
NOT = Invert Register/Memory	1131011w mod 010 r/m	2/6**	2/6**	b	h
STRING MANIPULATION					
CMPS = Compare Byte Word	1010011w			b	h
INS = Input Byte/Word from DX Port	0110110w	Clk Count Virtual 8086 Mode 129	15	9*/29**	b s/t, h, m
LODS = Load Byte/Word to AL/AX/EAX	1010110w		5	5*	b h
MOVS = Move Byte Word	1010010w		7	7**	b h
OUTS = Output Byte/Word to DX Port	0110111w	128	14	8*/28*	b s/t, h, m
SCAS = Scan Byte Word	1010111w		7*	7*	b h
STOS = Store Byte/Word from AL/AX/EX	1010101w		4*	4*	b h
XLAT = Translate String	11010111		5*	5*	h
REPEATED STRING MANIPULATION					
Repeated by Count in CX or ECX					
REPE CMPS = Compare String (Find Non-Match)	11110011 1010011w		5 + 9n**	5 + 9n**	b h

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Clk Count Virtual 8086 Mode	CLOCK COUNT		NOTES	
			Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
REPEATED STRING MANIPULATION (Continued)						
REPNE CMPS = Compare String (Find Match)	11110010 1010011w		5+9n*	5+9n**	b	h
REP INS = Input String	11110010 0110110w	†	13+6n*	7+6n*/ 27+6n*	b	s/t, h, m
REP LODS = Load String	11110010 1010110w		5+6n*	5+6n*	b	h
REP MOVS = Move String	11110010 1010010w		7+4n*	7+4n**	b	h
REP OUTS = Output String	11110010 0110111w	†	12+5n*	6+5n*/ 26+5n*	b	s/t, h, m
REPE SCAS = Scan String (Find Non-AL/AX/EAX)	11110011 1010111w		5+8n*	5+8n*	b	h
REPNE SCAS = Scan String (Find AL/AX/EAX)	11110010 1010111w		5+8n*	5+8n*	b	h
REP STOS = Store String	11110010 1010101w		5+5n*	5+5n*	b	h
BIT MANIPULATION						
BSF = Scan Bit Forward	00001111 10111100 mod,reg r/m		10+3n*	10+3n**	b	h
BSR = Scan Bit Reverse	00001111 10111101 mod,reg r/m		10+3n*	10+3n**	b	h
BT = Test Bit Register/Memory, Immediate	00001111 10111010 mod 100 r/m immed 8-bit data		3/6*	3/6*	b	h
Register/Memory, Register	00001111 10100011 mod,reg r/m		3/12*	3/12*	b	h
BTC = Test Bit and Complement Register/Memory, Immediate	00001111 10111010 mod 111 r/m immed 8-bit data		6/8*	6/8*	b	h
Register/Memory, Register	00001111 10111011 mod,reg r/m		6/13*	6/13*	b	h
BTR = Test Bit and Reset Register/Memory, Immediate	00001111 10111010 mod 110 r/m immed 8-bit data		6/8*	6/8*	b	h
Register/Memory, Register	00001111 10110011 mod,reg r/m		6/13*	6/13*	b	h
BTS = Test Bit and Set Register/Memory, Immediate	00001111 10111010 mod 101 r/m immed 8-bit data		6/8*	6/8*	b	h
Register/Memory, Register	00001111 10101011 mod,reg r/m		6/13*	6/13*	b	h
CONTROL TRANSFER						
CALL = Call Direct Within Segment	11101000 full displacement		7+m*	9+m*	b	r
Register/Memory Indirect Within Segment	11111111 mod 010 r/m		7+m*/10+m*	9+m/ 12+m*	b	h, r
Direct Intersegment	10011010 unsigned full offset, selector		17+m*	42+m*	b	j,k,r

NOTE:

† Clock count shown applies if I/O permission allows I/O to the port in virtual 8086 mode. If I/O bit map denies permission exception 13 fault occurs; refer to clock counts for INT 3 instruction.

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
CONTROL TRANSFER (Continued)					
Protected Mode Only (Direct Intersegment)					
			64 + m		h,j,k,r
Via Call Gate to Same Privilege Level					
Via Call Gate to Different Privilege Level, (No Parameters)			98 + m		h,j,k,r
Via Call Gate to Different Privilege Level, (x Parameters)					
From 286 Task to 286 TSS			106 + 8x + m		h,j,k,r
From 286 Task to 386™ SX CPU TSS			285		h,j,k,r
From 286 Task to Virtual 8086 Task (386 SX CPU TSS)			310		h,j,k,r
From 386 SX CPU Task to 286 TSS			229		h,j,k,r
From 386 SX CPU Task to 286 TSS			285		h,j,k,r
From 386 SX CPU Task to 386 SX CPU TSS			392		h,j,k,r
From 386 SX CPU Task to Virtual 8086 Task (386 SX CPU TSS)			309		h,j,k,r
Indirect Intersegment	1 1 1 1 1 1 1 1 mod 0 1 1 r/m	30 + m	46 + m	b	h,j,k,r
Protected Mode Only (Indirect Intersegment)					
Via Call Gate to Same Privilege Level			68 + m		h,j,k,r
Via Call Gate to Different Privilege Level, (No Parameters)			102 + m		h,j,k,r
Via Call Gate to Different Privilege Level, (x Parameters)					
From 286 Task to 286 TSS			110 + 8x + m		h,j,k,r
From 286 Task to 386 SX CPU TSS					h,j,k,r
From 286 Task to Virtual 8086 Task (386 SX CPU TSS)					h,j,k,r
From 386 SX CPU Task to 286 TSS					h,j,k,r
From 386 SX CPU Task to 386 SX CPU TSS			399		h,j,k,r
From 386 SX CPU Task to Virtual 8086 Task (386 SX CPU TSS)					h,j,k,r
JMP = Unconditional Jump					
Short	1 1 1 0 1 0 1 1 8-bit displacement	7 + m	7 + m		r
Direct within Segment	1 1 1 0 1 0 0 1 full displacement	7 + m	7 + m		r
Register/Memory Indirect within Segment	1 1 1 1 1 1 1 1 mod 1 0 0 r/m	9 + m/14 + m	9 + m/14 + m	b	h,r
Direct Intersegment	1 1 1 0 1 0 1 0 unsigned full offset, selector	16 + m	31 + m		j,k,r
Protected Mode Only (Direct Intersegment)					
Via Call Gate to Same Privilege Level			53 + m		h,j,k,r
From 286 Task to 286 TSS					h,j,k,r
From 286 Task to 386 SX CPU TSS					h,j,k,r
From 286 Task to Virtual 8086 Task (386 SX CPU TSS)					h,j,k,r
From 386 SX CPU Task to 286 TSS					h,j,k,r
From 386 SX CPU Task to 386 SX CPU TSS					h,j,k,r
From 386 SX CPU Task to Virtual 8086 Task (386 SX CPU TSS)			395		h,j,k,r
Indirect Intersegment	1 1 1 1 1 1 1 1 mod 1 0 1 r/m	17 + m	31 + m	b	h,j,k,r
Protected Mode Only (Indirect Intersegment)					
Via Call Gate to Same Privilege Level			49 + m		h,j,k,r
From 286 Task to 286 TSS					h,j,k,r
From 286 Task to 386 SX CPU TSS					h,j,k,r
From 286 Task to Virtual 8086 Task (386 SX CPU TSS)					h,j,k,r
From 386 SX CPU Task to 286 TSS					h,j,k,r
From 386 SX CPU Task to 386 SX CPU TSS			328		h,j,k,r
From 386 SX CPU Task to Virtual 8086 Task (386 SX CPU TSS)					h,j,k,r

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
CONTROL TRANSFER (Continued)					
RET = Return from CALL:					
Within Segment	1 1 0 0 0 0 1 1		12+m	b	g, h, r
Within Segment Adding Immediate to SP	1 1 0 0 0 0 1 0 16-bit displ.		12+m	b	g, h, r
Intersegment	1 1 0 0 1 0 1 1		36+m	b	g, h, j, k, r
Intersegment Adding Immediate to SP	1 1 0 0 1 0 1 0 16-bit displ.		36+m	b	g, h, j, k, r
Protected Mode Only (RET): to Different Privilege Level					
Intersegment			72		h, j, k, r
Intersegment Adding Immediate to SP			72		h, j, k, r
CONDITIONAL JUMPS					
NOTE: Times Are Jump "Taken or Not Taken"					
JO = Jump on Overflow					
8-Bit Displacement	0 1 1 1 0 0 0 0 8-bit displ.	7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 full displacement	7+m or 3	7+m or 3		r
JNO = Jump on Not Overflow					
8-Bit Displacement	0 1 1 1 0 0 0 1 8-bit displ.	7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 full displacement	7+m or 3	7+m or 3		r
JB/JNAE = Jump on Below/Not Above or Equal					
8-Bit Displacement	0 1 1 1 0 0 1 0 8-bit displ.	7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 0 full displacement	7+m or 3	7+m or 3		r
JNB/JAE = Jump on Not Below/Above or Equal					
8-Bit Displacement	0 1 1 1 0 0 1 1 8-bit displ.	7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 full displacement	7+m or 3	7+m or 3		r
JE/JZ = Jump on Equal/Zero					
8-Bit Displacement	0 1 1 1 0 1 0 0 8-bit displ.	7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 full displacement	7+m or 3	7+m or 3		r
JNE/JNZ = Jump on Not Equal/Not Zero					
8-Bit Displacement	0 1 1 1 0 1 0 1 8-bit displ.	7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 1 full displacement	7+m or 3	7+m or 3		r
JBE/JNA = Jump on Below or Equal/Not Above					
8-Bit Displacement	0 1 1 1 0 1 1 0 8-bit displ.	7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 0 full displacement	7+m or 3	7+m or 3		r
JNBE/JA = Jump on Not Below or Equal/Above					
8-Bit Displacement	0 1 1 1 0 1 1 1 8-bit displ.	7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 full displacement	7+m or 3	7+m or 3		r
JS = Jump on Sign					
8-Bit Displacement	0 1 1 1 1 0 0 0 8-bit displ.	7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 1 0 0 0 full displacement	7+m or 3	7+m or 3		r

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
CONDITIONAL JUMPS (Continued)					
JNS = Jump on Not Sign					
8-Bit Displacement	0 111 1001 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 10001001 full displacement	7+m or 3	7+m or 3		r
JJP/JPE = Jump on Parity/Parity Even					
8-Bit Displacement	0 111 1010 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 10001010 full displacement	7+m or 3	7+m or 3		r
JNP/JPO = Jump on Not Parity/Parity Odd					
8-Bit Displacement	0 111 1011 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 10001011 full displacement	7+m or 3	7+m or 3		r
JL/JNGE = Jump on Less/Not Greater or Equal					
8-Bit Displacement	0 111 1100 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 10001100 full displacement	7+m or 3	7+m or 3		r
JNL/JGE = Jump on Not Less/Greater or Equal					
8-Bit Displacement	0 111 1101 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 10001101 full displacement	7+m or 3	7+m or 3		r
JLE/JNG = Jump on Less or Equal/Not Greater					
8-Bit Displacement	0 111 1110 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 10001110 full displacement	7+m or 3	7+m or 3		r
JNLE/JG = Jump on Not Less or Equal/Greater					
8-Bit Displacement	0 111 1111 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 10001111 full displacement	7+m or 3	7+m or 3		r
JCXZ = Jump on CX Zero					
	11100011 8-bit displ	9+m or 5	9+m or 5		r
JECXZ = Jump on ECX Zero					
	11100011 8-bit displ	9+m or 5	9+m or 5		r
(Address Size Prefix Differentiates JCXZ from JECXZ)					
LOOP = Loop CX Times					
	11100010 8-bit displ	11+m	11+m		r
LOOPZ/LOOPE = Loop with Zero/Equal					
	11100001 8-bit displ	11+m	11+m		r
LOOPNZ/LOOPNE = Loop While Not Zero					
	11100000 8-bit displ	11+m	11+m		r
CONDITIONAL BYTE SET					
NOTE: Times Are Register/Memory					
SETO = Set Byte on Overflow					
To Register/Memory	00001111 10010000 mod 0 00 r/m	4/5*	4/5*		h
SETNO = Set Byte on Not Overflow					
To Register/Memory	00001111 10010001 mod 0 00 r/m	4/5*	4/5*		h
SETB/SETNAE = Set Byte on Below/Not Above or Equal					
To Register/Memory	00001111 10010010 mod 0 00 r/m	4/5*	4/5*		h

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
CONDITIONAL BYTE SET (Continued)					
SETNB = Set Byte on Not-Below/Above or Equal					
To Register/Memory	00001111 10010011 mod000 r/m	4/5*	4/5*		h
SETE/SETZ = Set Byte on Equal/Zero					
To Register/Memory	00001111 10010100 mod000 r/m	4/5*	4/5*		h
SETNE/SETNZ = Set Byte on Not Equal/Not Zero					
To Register/Memory	00001111 10010101 mod000 r/m	4/5*	4/5*		h
SETBE/SETNA = Set Byte on Below or Equal/Not Above					
To Register/Memory	00001111 10010110 mod000 r/m	4/5*	4/5*		h
SETNBE/SETA = Set Byte on Not Below or Equal/Above					
To Register/Memory	00001111 10010111 mod000 r/m	4/5*	4/5*		h
SETS = Set Byte on Sign					
To Register/Memory	00001111 10011000 mod000 r/m	4/5*	4/5*		h
SETNS = Set Byte on Not Sign					
To Register/Memory	00001111 10011001 mod000 r/m	4/5*	4/5*		h
SETP/SETPE = Set Byte on Parity/Parity Even					
To Register/Memory	00001111 10011010 mod000 r/m	4/5*	4/5*		h
SETNP/SETPO = Set Byte on Not Parity/Parity Odd					
To Register/Memory	00001111 10011011 mod000 r/m	4/5*	4/5*		h
SETL/SETNGE = Set Byte on Less/Not Greater or Equal					
To Register/Memory	00001111 10011100 mod000 r/m	4/5*	4/5*		h
SETNL/SETGE = Set Byte on Not Less/Greater or Equal					
To Register/Memory	00001111 01111101 mod000 r/m	4/5*	4/5*		h
SETLE/SETNG = Set Byte on Less or Equal/Not Greater					
To Register/Memory	00001111 10011110 mod000 r/m	4/5*	4/5*		h
SETNLE/SETG = Set Byte on Not Less or Equal/Greater					
To Register/Memory	00001111 10011111 mod000 r/m	4/5*	4/5*		h
ENTER = Enter Procedure	11001000 16-bit displacement, 8-bit level				
L = 0		10	10	b	h
L = 1		14	14	b	h
L > 1		17 +	17 +	b	h
		8(n - 1)	8(n - 1)		
LEAVE = Leave Procedure	11001001	4	4	b	h

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
INTERRUPT INSTRUCTIONS					
INT = Interrupt:					
Type Specified	11001101 type	37		b	
Type 3	11001100	33		b	
INTO = Interrupt 4 if Overflow Flag Set					
	11001110				
If OF = 1		35		b, e	
If OF = 0		3	3	b, e	
Bound = Interrupt 5 if Detect Value Out of Range					
	01100010 mod reg r/m				
If Out of Range		44		b, e	e, g, h, j, k, r
If In Range		10	10	b, e	e, g, h, j, k, r
Protected Mode Only (INT)					
INT: Type Specified					
Via Interrupt or Trap Gate					
Via Interrupt or Trap Gate to Same Privilege Level					
to Different Privilege Level					
From 286 Task to 286 TSS via Task Gate					
From 286 Task to 386™ SX CPU TSS via Task Gate					
From 286 Task to virt 8086 md via Task Gate					
From 386™ SX CPU Task to 286 TSS via Task Gate					
From 386™ SX CPU Task to 386™ SX CPU TSS via Task Gate					
From 386™ SX CPU Task to virt 8086 md via Task Gate					
From virt 8086 md to 286 TSS via Task Gate					
From virt 8086 md to 386™ SX CPU TSS via Task Gate					
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate					
			71		g, j, k, r
			111		g, j, k, r
			438		g, j, k, r
			465		g, j, k, r
			382		g, j, k, r
			440		g, j, k, r
			467		g, j, k, r
			384		g, j, k, r
			445		g, j, k, r
			472		g, j, k, r
			275		
INT: TYPE 3					
Via Interrupt or Trap Gate to Same Privilege Level					
to Different Privilege Level					
From 286 Task to 286 TSS via Task Gate					
From 286 Task to 386™ SX CPU TSS via Task Gate					
From 286 Task to Virt 8086 md via Task Gate					
From 386™ SX CPU Task to 286 TSS via Task Gate					
From 386™ SX CPU Task to 386™ SX CPU TSS via Task Gate					
From 386™ SX CPU Task to Virt 8086 md via Task Gate					
From virt 8086 md to 286 TSS via Task Gate					
From virt 8086 md to 386™ SX CPU TSS via Task Gate					
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate					
			71		g, j, k, r
			111		g, j, k, r
			384		g, j, k, r
			411		g, j, k, r
			328		g, j, k, r
			389		g, j, k, r
			416		g, j, k, r
			223		
INTO:					
Via Interrupt or Trap Gate to Same Privilege Level					
to Different Privilege Level					
From 286 Task to 286 TSS via Task Gate					
From 286 Task to 386™ SX CPU TSS via Task Gate					
From 286 Task to virt 8086 md via Task Gate					
From 386™ SX CPU Task to 286 TSS via Task Gate					
From 386™ SX CPU Task to 386™ SX CPU TSS via Task Gate					
From 386™ SX CPU Task to virt 8086 md via Task Gate					
From virt 8086 md to 286 TSS via Task Gate					
From virt 8086 md to 386™ SX CPU TSS via Task Gate					
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate					
			71		g, j, k, r
			111		g, j, k, r
			384		g, j, k, r
			411		g, j, k, r
			328		g, j, k, r
			386 DX		g, j, k, r
			413		g, j, k, r
			329		g, j, k, r
			391		g, j, k, r
			418		g, j, k, r
			223		

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
INTERRUPT INSTRUCTIONS (Continued)					
BOUND:					
	Via Interrupt or Trap Gate to Same Privilege Level		71		g, j, k, r
	Via Interrupt or Trap Gate to Different Privilege Level		111		g, j, k, r
	From 286 Task to 286 TSS via Task Gate		358		g, j, k, r
	From 286 Task to 386™ SX CPU TSS via Task Gate		389		g, j, k, r
	From 286 Task to virt 8086 Mode via Task Gate		335		g, j, k, r
	From 386 SX CPU Task to 286 TSS via Task Gate		368		g, j, k, r
	From 386 SX CPU Task to 386 SX CPU TSS via Task Gate		398		g, j, k, r
	From 386 SX CPU Task to virt 8086 Mode via Task Gate		347		g, j, k, r
	From virt 8086 Mode to 286 TSS via Task Gate		368		g, j, k, r
	From virt 8086 Mode to 386 SX CPU TSS via Task Gate		398		g, j, k, r
	From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate		223		
INTERRUPT RETURN					
IRET = Interrupt Return	11001111		24		g, h, j, k, r
Protected Mode Only (IRET)					
	To the Same Privilege Level (within task)		42		g, h, j, k, r
	To Different Privilege Level (within task)		86		g, h, j, k, r
	From 286 Task to 286 TSS		285		h, j, k, r
	From 286 Task to 386 SX CPU TSS		318		h, j, k, r
	From 286 Task to Virtual 8086 Task		267		h, j, k, r
	From 286 Task to Virtual 8086 Mode (within task)		113		
	From 386 SX CPU Task to 286 TSS		324		h, j, k, r
	From 386 SX CPU Task to 386 SX CPU TSS		328		h, j, k, r
	From 386 SX CPU Task to Virtual 8086 Task		377		h, j, k, r
	From 386 SX CPU Task to Virtual 8086 Mode (within task)		113		
PROCESSOR CONTROL					
HLT = HALT	11110100		5	5	I
MOV = Move to and From Control/Debug/Test Registers					
CR0/CR2/CR3 from register	00001111 00100010 11 eee reg	10/4/5	10/4/5		I
Register From CR0-3	00001111 00100000 11 eee reg	6	6		I
DR0-3 From Register	00001111 00100011 11 eee reg	22	22		I
DR6-7 From Register	00001111 00100011 11 eee reg	16	16		I
Register from DR6-7	00001111 00100001 11 eee reg	14	14		I
Register from DR0-3	00001111 00100001 11 eee reg	22	22		I
TR6-7 from Register	00001111 00100110 11 eee reg	12	12		I
Register from TR6-7	00001111 00100100 11 eee reg	12	12		I
NOP = No Operation	10010000		3	3	
WAIT = Wait until BUSY # pin is negated	10011011		6	6	

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES				
		Real Address Mode or Virtual Address 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual Address 8086 Mode	Protected Virtual Address Mode			
PROCESSOR EXTENSION INSTRUCTIONS								
Processor Extension Escape	<table border="1"> <tr> <td>1 1 0 1 1 T T T</td> <td>mod L L L</td> <td>r/m</td> </tr> </table> <p>TTT and LLL bits are opcode information for coprocessor.</p>	1 1 0 1 1 T T T	mod L L L	r/m	See 387SX data sheet for clock counts		h	
1 1 0 1 1 T T T	mod L L L	r/m						
PREFIX BYTES								
Address Size Prefix	<table border="1"><tr><td>0 1 1 0 0 1 1 1</td></tr></table>	0 1 1 0 0 1 1 1	0	0				
0 1 1 0 0 1 1 1								
LOCK = Bus Lock Prefix	<table border="1"><tr><td>1 1 1 1 0 0 0 0</td></tr></table>	1 1 1 1 0 0 0 0	0	0		m		
1 1 1 1 0 0 0 0								
Operand Size Prefix	<table border="1"><tr><td>0 1 1 0 0 1 1 0</td></tr></table>	0 1 1 0 0 1 1 0	0	0				
0 1 1 0 0 1 1 0								
Segment Override Prefix								
CS:	<table border="1"><tr><td>0 0 1 0 1 1 1 0</td></tr></table>	0 0 1 0 1 1 1 0	0	0				
0 0 1 0 1 1 1 0								
DS:	<table border="1"><tr><td>0 0 1 1 1 1 1 0</td></tr></table>	0 0 1 1 1 1 1 0	0	0				
0 0 1 1 1 1 1 0								
ES:	<table border="1"><tr><td>0 0 1 0 0 1 1 0</td></tr></table>	0 0 1 0 0 1 1 0	0	0				
0 0 1 0 0 1 1 0								
FS:	<table border="1"><tr><td>0 1 1 0 0 1 0 0</td></tr></table>	0 1 1 0 0 1 0 0	0	0				
0 1 1 0 0 1 0 0								
GS:	<table border="1"><tr><td>0 1 1 0 0 1 0 1</td></tr></table>	0 1 1 0 0 1 0 1	0	0				
0 1 1 0 0 1 0 1								
SS:	<table border="1"><tr><td>0 0 1 1 0 1 1 0</td></tr></table>	0 0 1 1 0 1 1 0	0	0				
0 0 1 1 0 1 1 0								
PROTECTION CONTROL								
ARPL = Adjust Requested Privilege Level								
From Register/Memory	<table border="1"> <tr> <td>0 1 1 0 0 0 1 1</td> <td>mod reg</td> <td>r/m</td> </tr> </table>	0 1 1 0 0 0 1 1	mod reg	r/m	N/A	20/21**	a h	
0 1 1 0 0 0 1 1	mod reg	r/m						
LAR = Load Access Rights								
From Register/Memory	<table border="1"> <tr> <td>0 0 0 0 1 1 1 1</td> <td>0 0 0 0 0 0 1 0</td> <td>mod reg</td> <td>r/m</td> </tr> </table>	0 0 0 0 1 1 1 1	0 0 0 0 0 0 1 0	mod reg	r/m	N/A	15/16*	a g, h, j, p
0 0 0 0 1 1 1 1	0 0 0 0 0 0 1 0	mod reg	r/m					
LGDT = Load Global Descriptor								
Table Register	<table border="1"> <tr> <td>0 0 0 0 1 1 1 1</td> <td>0 0 0 0 0 0 0 1</td> <td>mod 0 1 0</td> <td>r/m</td> </tr> </table>	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 1 0	r/m	11*	11*	b, c h, l
0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 1 0	r/m					
LIDT = Load Interrupt Descriptor								
Table Register	<table border="1"> <tr> <td>0 0 0 0 1 1 1 1</td> <td>0 0 0 0 0 0 0 1</td> <td>mod 0 1 1</td> <td>r/m</td> </tr> </table>	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 1 1	r/m	11*	11*	b, c h, l
0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 1 1	r/m					
LLDT = Load Local Descriptor								
Table Register to Register/Memory	<table border="1"> <tr> <td>0 0 0 0 1 1 1 1</td> <td>0 0 0 0 0 0 0 0</td> <td>mod 0 1 0</td> <td>r/m</td> </tr> </table>	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 1 0	r/m	N/A	20/24*	a g, h, j, l
0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 1 0	r/m					
LMSW = Load Machine Status Word								
From Register/Memory	<table border="1"> <tr> <td>0 0 0 0 1 1 1 1</td> <td>0 0 0 0 0 0 0 1</td> <td>mod 1 1 0</td> <td>r/m</td> </tr> </table>	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 1 1 0	r/m	10/13	10/13*	b, c h, l
0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 1 1 0	r/m					
LSL = Load Segment Limit								
From Register/Memory	<table border="1"> <tr> <td>0 0 0 0 1 1 1 1</td> <td>0 0 0 0 0 0 1 1</td> <td>mod reg</td> <td>r/m</td> </tr> </table>	0 0 0 0 1 1 1 1	0 0 0 0 0 0 1 1	mod reg	r/m	N/A	20/21*	a g, h, j, p
0 0 0 0 1 1 1 1	0 0 0 0 0 0 1 1	mod reg	r/m					
Byte-Granular Limit		N/A	25/26*	a	g, h, j, p			
LTR = Load Task Register								
From Register/Memory	<table border="1"> <tr> <td>0 0 0 0 1 1 1 1</td> <td>0 0 0 0 0 0 0 0</td> <td>mod 0 0 1</td> <td>r/m</td> </tr> </table>	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 0 1	r/m	N/A	23/27*	a g, h, j, l
0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 0 1	r/m					
SGDT = Store Global Descriptor								
Table Register	<table border="1"> <tr> <td>0 0 0 0 1 1 1 1</td> <td>0 0 0 0 0 0 0 1</td> <td>mod 0 0 0</td> <td>r/m</td> </tr> </table>	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 0 0	r/m	9*	9*	b, c h
0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 0 0	r/m					
SIDT = Store Interrupt Descriptor								
Table Register	<table border="1"> <tr> <td>0 0 0 0 1 1 1 1</td> <td>0 0 0 0 0 0 0 1</td> <td>mod 0 0 1</td> <td>r/m</td> </tr> </table>	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 0 1	r/m	9*	9*	b, c h
0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 0 1	r/m					
SLDT = Store Local Descriptor Table Register								
To Register/Memory	<table border="1"> <tr> <td>0 0 0 0 1 1 1 1</td> <td>0 0 0 0 0 0 0 0</td> <td>mod 0 0 0</td> <td>r/m</td> </tr> </table>	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 0 0	r/m	N/A	2/2*	a h
0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 0 0	r/m					

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES					
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode				
PROTECTION CONTROL (Continued)									
SMSW = Store Machine Status Word	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>00001111</td><td>00000001</td><td>mod 100</td><td>r/m</td></tr></table>	00001111	00000001	mod 100	r/m	2/2*	2/2*	b, c	h, l
00001111	00000001	mod 100	r/m						
STR = Store Task Register To Register/Memory	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>00001111</td><td>00000000</td><td>mod 001</td><td>r/m</td></tr></table>	00001111	00000000	mod 001	r/m	N/A	2/2*	a	h
00001111	00000000	mod 001	r/m						
VERR = Verify Read Access Register/Memory	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>00001111</td><td>00000000</td><td>mod 100</td><td>r/m</td></tr></table>	00001111	00000000	mod 100	r/m	N/A	10/11*	a	g, h, j, p
00001111	00000000	mod 100	r/m						
VERW = Verify Write Access	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>00001111</td><td>00000000</td><td>mod 101</td><td>r/m</td></tr></table>	00001111	00000000	mod 101	r/m	N/A	15/16*	a	g, h, j, p
00001111	00000000	mod 101	r/m						

INSTRUCTION NOTES FOR TABLE 9-1

Notes a through c apply to Real Address Mode only:

- a. This is a Protected Mode instruction. Attempted execution in Real Mode will result in exception 6 (invalid opcode).
- b. Exception 13 fault (general protection) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum CS, DS, ES, FS or GS limit, FFFFH. Exception 12 fault (stack segment limit violation or not present) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.
- c. This instruction may be executed in Real Mode. In Real Mode, its purpose is primarily to initialize the CPU for Protected Mode.

Notes d through g apply to Real Address Mode and Protected Virtual Address Mode:

d. The 386 SX CPU uses an early-out multiply algorithm. The actual number of clocks depends on the position of the most significant bit in the operand (multiplier).

Clock counts given are minimum to maximum. To calculate actual clocks use the following formula:

$$\text{Actual Clock} = \text{if } m < > 0 \text{ then } \max(\lceil \log_2 [m] \rceil, 3) + b \text{ clocks}$$

$$\text{if } m = 0 \text{ then } 3 + b \text{ clocks}$$

- In this formula, m is the multiplier, and
- b = 9 for register to register,
- b = 12 for memory to register,
- b = 10 for register with immediate to register,
- b = 11 for memory with immediate to register.

- e. An exception may occur, depending on the value of the operand.
- f. LOCK# is automatically asserted, regardless of the presence or absence of the LOCK# prefix.
- g. LOCK# is asserted during descriptor table accesses.

Notes h through r apply to Protected Virtual Address Mode only:

- h. Exception 13 fault (general protection violation) will occur if the memory operand in CS, DS, ES, FS or GS cannot be used due to either a segment limit violation or access rights violation. If a stack limit is violated, an exception 12 (stack segment limit violation or not present) occurs.
- i. For segment load operations, the CPL, RPL, and DPL must agree with the privilege rules to avoid an exception 13 fault (general protection violation). The segment's descriptor must indicate "present" or exception 11 (CS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 (stack segment limit violation or not present) occurs.
- j. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert LOCK# to maintain descriptor integrity in multiprocessor systems.
- k. JMP, CALL, INT, RET and IRET instructions referring to another code segment will cause an exception 13 (general protection violation) if an applicable privilege rule is violated.
- l. An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).
- m. An exception 13 fault occurs if CPL is greater than IOPL.
- n. The IF bit of the flag register is not updated if CPL is greater than IOPL. The IOPL and VM fields of the flag register are updated only if CPL = 0.
- o. The PE bit of the MSW (CR0) cannot be reset by this instruction. Use MOV into CR0 if desiring to reset the PE bit.
- p. Any violation of privilege rules as applied to the selector operand does not cause a protection exception; rather, the zero flag is cleared.
- q. If the coprocessor's memory operand violates a segment limit or segment access rights, an exception 13 fault (general protection exception) will occur before the ESC instruction is executed. An exception 12 fault (stack segment limit violation or not present) will occur if the stack limit is violated by the operand's starting address.
- r. The destination of a JMP, CALL, INT, RET or IRET must be in the defined limit of a code segment or an exception 13 fault (general protection violation) will occur.
- s/t. The instruction will execute in s clocks if CPL ≤ IOPL. If CPL > IOPL, the instruction will take t clocks.

9.2 INSTRUCTION ENCODING

9.2.1 Overview

All instruction encodings are subsets of the general instruction format shown in Figure 8-1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the “mod r/m” byte and “scaled index” byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain

encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 9-1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 9-2 is a complete list of all fields appearing in the instruction set. Further ahead, following Table 9-2, are detailed tables for each field.

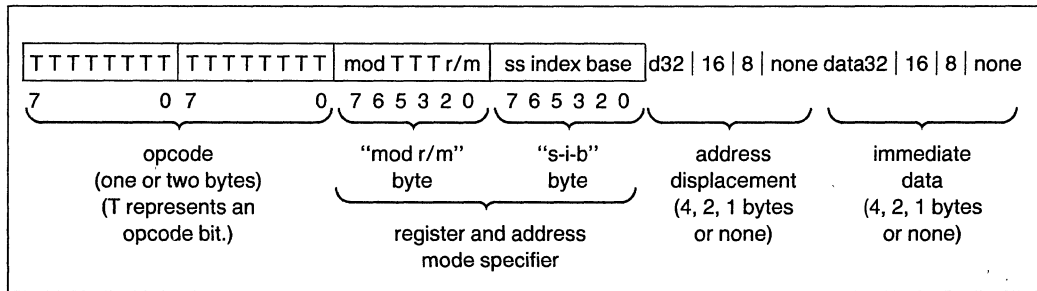


Figure 9-1. General Instruction Format

Table 9-2. Fields within Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 Bits)	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field Must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
ttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

Note: Table 9-1 shows encoding of individual instructions.

9.2.2 32-Bit Extensions of the Instruction Set

With the 386 SX CPU, the 8086/80186/80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 bits or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bits or 16 bits) for both operands and effective addresses when executing that code segment. In the Real Address Mode or Virtual 8086 Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the 386 SX CPU when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the opcode bytes. The presence of the Operand Size Prefix and the Effective Address Prefix will toggle the operand size or the effective address size, respectively, to the value "opposite" from the Default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. As another example, if the default effective address size is 16 bits, presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.

These 32-bit extensions are available in all modes, including the Real Address Mode or the Virtual 8086 Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

9.2.3 Encoding of Instruction Fields

Within the instruction are several fields indicating register selection, addressing mode and so on. The exact encodings of these fields are defined immediately ahead.

9.2.3.1 ENCODING OF OPERAND LENGTH (w) FIELD

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size During 16-Bit Data Operations	Operand Size During 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

9.2.3.2 ENCODING OF THE GENERAL REGISTER (reg) FIELD

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the "mod r/m" byte, or as the r/m field of the "mod r/m" byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-Bit Data Operations	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
101	SI	ESI
101	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations:		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-Bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

9.2.3.3 ENCODING OF THE SEGMENT REGISTER (sreg) FIELD

The sreg field in certain instructions is a 2-bit field allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the 386 SX CPU FS and GS segment registers to be specified.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

9.2.3.4 ENCODING OF ADDRESS MODE

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the “mod r/m” byte, and a second byte of addressing information, the “s-i-b” (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the “mod r/m” byte has r/m = 100 and mod = 00, 01 or 10. When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the “mod r/m” byte, also contains three bits (shown as TTT in Figure 8-1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the “mod r/m” byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the “mod r/m” byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following three pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Encoding of 16-bit Address Mode with “mod r/m” Byte

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	SS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by r/m During 16-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode with “mod r/m” byte (no “s-i-b” byte present):

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by reg or r/m during 16-Bit Data Operations:		
mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by reg or r/m during 32-Bit Data Operations:		
mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode (“mod r/m” byte and “s-i-b” byte present):

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

****IMPORTANT NOTE:**

When index field is 100, indicating “no index register,” then ss field **MUST** equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

NOTE:

Mod field in “mod r/m” byte; ss, index, base fields in “s-i-b” byte.

9.2.3.5 ENCODING OF OPERATION DIRECTION (d) FIELD

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory <- - Register "reg" Field Indicates Source Operand; "mod r/m" or "mod ss index base" Indicates Destination Operand
1	Register <- - Register/Memory "reg" Field Indicates Destination Operand; "mod r/m" or "mod ss index base" Indicates Source Operand

9.2.3.6 ENCODING OF SIGN-EXTEND (s) FIELD

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16 32
0	None	None
1	Sign-Extend Data8 to Fill 16-Bit or 32-Bit Destination	None

9.2.3.7 ENCODING OF CONDITIONAL TEST (tttn) FIELD

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n = 0) or its negation (n = 1), and tt giving the condition to test.

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

9.2.3.8 ENCODING OF CONTROL OR DEBUG OR TEST REGISTER (eee) FIELD

For the loading and storing of the Control, Debug and Test registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3
Do not use any other encoding	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

When Interpreted as Test Register Field

eee Code	Reg Name
110	TR6
111	TR7
Do not use any other encoding	

DATA SHEET REVISION REVIEW

The following list represents key differences between this and the -001 version of the 386™ SX microprocessor data sheet. Please review this summary carefully.

The section significantly revised since version -002 is:

Section 1.0 Figure 1.1 was modified to also give pin names. Table 1.1 was modified to list pin names in alphabetical order.

The sections significantly revised since version -003 are:

Section 7.3 Table 7.3 modified to show new I_{CC} values at 16 MHz and 20 MHz.

Section 7.4 Add 20 MHz A.C. Specifications in Table 7.5. Modified capacitive derating information in Tables 7.8 through 7.11. Modified typical I_{CC} vs. frequency in Table 7.12.

387™ SX MATH COPROCESSOR

- Interfaces with 386™ SX Microprocessor
- Expands 386 SX CPU Data Types to Include 32-, 64-, 80-Bit Floating Point, 32-, 64-Bit Integers and 18-Digit BCD Operands
- High Performance 80-Bit Internal Architecture
- Two to Three Times 8087/80287 Performance at Equivalent Clock Speed
- Implements ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic
- Fully compatible with the 387™ Math Coprocessor. Implements all 387 NPX architectural enhancements over 8087 and 80287.
- Upward Object-Code Compatible from 8087 and 80287
- Directly Extends 386 SX CPU Instruction Set to Trigonometric, Logarithmic, Exponential, and Arithmetic Instructions for All Data Types
- Full-Range Transcendental Operations for SINE, COSINE, TANGENT, ARCTANGENT, and LOGARITHM.
- Operates Independently of Real, Protected, and Virtual-8086 Modes of the 386 SX Microprocessor
- Eight 80-Bit Numeric Registers, Usable as Individually Addressable General Registers or as a Register Stack
- Available in a 68-pin PLCC Package (see Packaging Specs: Order # 231369)

The Intel 387™ SX Math CoProcessor is an extension to the Intel 386™ microprocessor architecture. The combination of the 387 SX with the 386™ SX Microprocessor dramatically increases the processing speed of computer application software which utilizes mathematical operations. This makes an ideal computer workstation platform for applications such as financial modeling and spreadsheets, CAD/CAM, or graphics.

The 387 SX Math CoProcessor adds over seventy mnemonics to the 386 SX Microprocessor instruction set. Specific 387 SX math operations include logarithmic, arithmetic, exponential, and trigonometric functions. The 387 SX supports integer, extended integer, floating point and BCD data formats, and fully conforms to the ANSI/IEEE floating point standard.

The 387 SX Math CoProcessor is object code compatible with the 387™ DX and upward object code compatible from the 80287 and 8087 Math Coprocessors. The 387 SX is manufactured with Intel's CHMOS III technology and packaged in a 68-pin PLCC package. A low power consumption option allows use in laptop or portable applications.

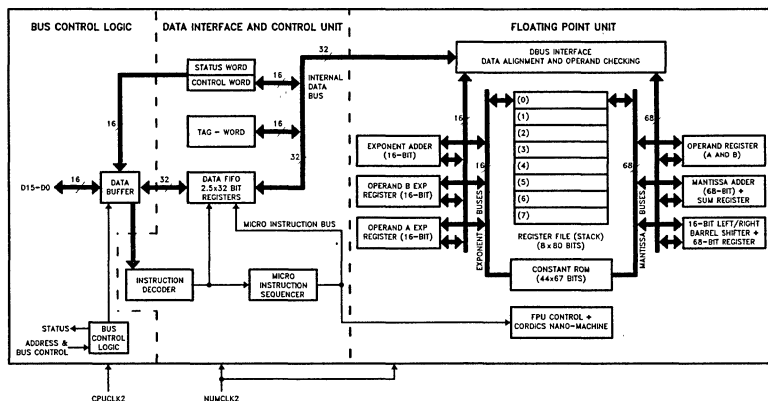


Figure 0-1. Block Diagram

240225-1

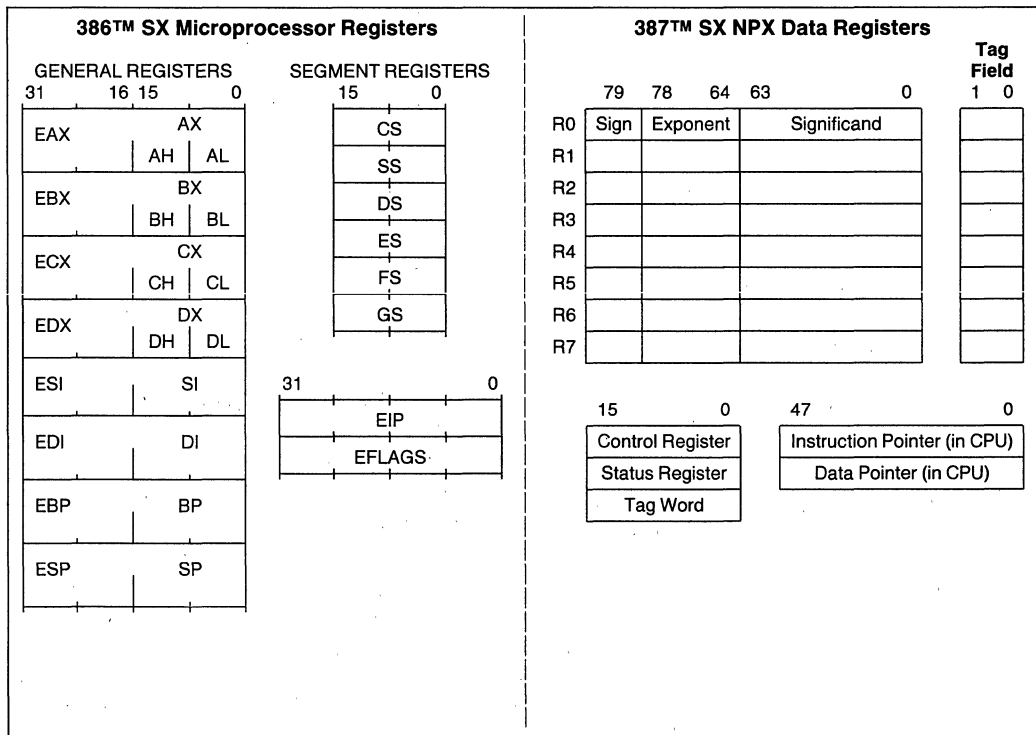


Figure 1-1. 386™ SX Microprocessor and 387™ SX Math Coprocessor Register Set

1.0 FUNCTIONAL DESCRIPTION

The 387™ SX Math Coprocessor Extension (NPX) provides arithmetic instructions for a variety of numeric data types. It also executes numerous built-in transcendental functions (e.g. tangent, sine, cosine, and log functions). The 387 SX NPX effectively extends the register and instruction set of its CPU for existing data types and adds several new data types as well. Figure 1-1 shows the model of registers visible to 386™ SX Microprocessor and 387 SX Math Coprocessor applications programs. Essentially, the 387 SX Math Coprocessor can be treated as an additional resource or an extension to the 386 SX Microprocessor. The 386 SX Microprocessor together with a 387 SX NPX can be used as a single unified system, the 386 SX Microprocessor and 387 SX Math Coprocessor.

The 387 SX Numerics Coprocessor Extension works the same whether the CPU is executing in real-address mode, protected mode, or virtual-8086 mode. All references to memory for numerics data or status information are performed by the CPU, and therefore obey the memory-management and protection rules of the CPU mode currently in effect. The 387 SX Numerics Coprocessor Extension merely operates on instructions and values passed to it by the

CPU and therefore is not sensitive to the processing mode of the CPU.

In real-address mode and virtual-8086 mode, the 386 SX Microprocessor and 387 SX Math Coprocessor is completely upward compatible with software for the 8086/8087 and 80286/80287 real-address mode systems.

In protected mode, the 386 SX Microprocessor and 387 SX Math Coprocessor is completely upward compatible with software for the 80286/80287 protected mode system.

In all modes, the 386 SX Microprocessor and 387 SX Math Coprocessor is completely compatible with software for the 386™ Microprocessor/387™ Math Coprocessor system.

The only differences of operation that may appear when 8086/8087 programs are ported to the protected-mode 386 SX Microprocessor and 387 SX Math Coprocessor system (*not* using virtual-8086 mode) is in the format of operands for the administrative instructions FLDENV, FSTENV, FRSTOR, and FSAVE. These instruction are normally used only by exception handlers and operating systems, not by applications programs.

2.0 PROGRAMMING INTERFACE

The 387 SX NPX adds to an 386 SX Microprocessor system additional data types, registers, instructions, and interrupts specifically designed to facilitate high-speed numerics processing. To use the 387 SX NPX requires no special programming tools, because all new instructions and data types are directly supported by the assembler and compilers for high-level languages. All 386 Microprocessor development tools that support 387 NPX programs can also be used to develop software for the 386 SX Microprocessor and 387 SX Math Coprocessor. All 8086/8088 development tools that support the 8087 can also be used to develop software for the 386 SX Microprocessor and 387 SX Math Coprocessor in real-address mode or virtual-8086 mode. All 80286 development tools that support the 80287 can also be used to develop software for the 386 SX Microprocessor and 387 SX Math Coprocessor.

The 387 SX NPX supports all 387 NPX instructions. The 386 SX Microprocessor and 387 SX Math Coprocessor supports all the same programs and gives the same results as an 386 Microprocessor and 387 Math Coprocessor.

All communication between the CPU and the NPX is transparent to applications software. The CPU automatically controls the NPX whenever a numerics instruction is executed. All physical memory and virtual memory of the CPU are available for storage of the instructions and operands of programs that use the NPX. All memory addressing modes, including use of displacement, base register, index register, and scaling, are available for addressing numerics operands.

Section 7 at the end of this data sheet lists by class the instructions that the 387 SX NPX adds to the instruction set of an 386 SX Microprocessor system.

2.1 Data Types

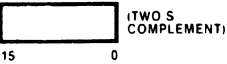
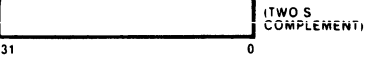
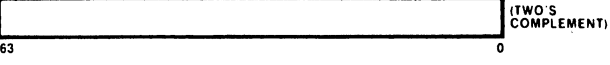
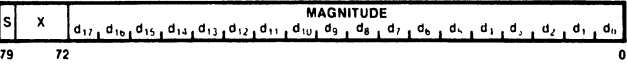
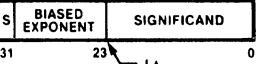
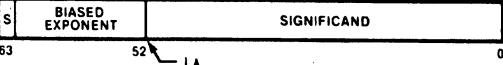
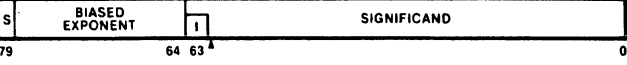
Table 2-1 lists the seven data types that the NPX supports and presents the format for each type. Operands are stored in memory with the least significant digit at the lowest memory address. Programs retrieve these values by generating the lowest address. For maximum system performance, all operands should start at physical-memory addresses that correspond to the word size of the CPU; operands may begin at any other addresses, but will require extra memory cycles to access the entire operand.

Internally, the NPX holds all numbers in the extended-precision real format. Instructions that load operands from memory automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating-point numbers, or 18-digit packed BCD numbers into extended-precision real format. Instructions that store operands in memory perform the inverse type conversion.

2.2 Numeric Operands

A typical NPX instruction accepts one or two operands and produces one (or sometimes two) results. In two-operand instructions, one operand is the contents of an NPX register, while the other may be a memory location. The operands of some instructions are predefined; for example, FSQRT always takes the square root of the number in the top stack element.

Table 2-1. 387™ SX NPX Data Type Representation in Memory

Data Formats	Range	Precision	Most Significant Byte = HIGHEST ADDRESSED BYTE											
			7	0	7	0	7	0	7	0	7	0	7	0
Word Integer	$\pm 10^4$	16 Bits												
Short Integer	$\pm 10^9$	32 Bits												
Long Integer	$\pm 10^{18}$	64 Bits												
Packed BCD	$\pm 10^{18}$	18 Digits												
Single Precision	$\pm 10^{\pm 38}$	24 Bits												
Double Precision	$\pm 10^{\pm 308}$	53 Bits												
Extended Precision	$\pm 10^{\pm 4932}$	64 Bits												

240225-2

NOTES:

- (1) S = Sign bit (0 = positive, 1 = negative)
- (2) d_n = Decimal digit (two per byte)
- (3) X = Bits have no significance; NPX ignores when loading, zeros when storing
- (4) Δ = Position of implicit binary point
- (5) I = Integer bit of significand; stored in temporary real, implicit in single and double precision
- (6) Exponent Bias (normalized values):
 Single: 127 (7FH)
 Double: 1023 (3FFH)
 Extended REal: 16383 (3FFFH)
- (7) Packed BCD: $(-1)^S (D_{17}..D_0)$
- (8) Real: $(-1)^S (2^E\text{-BIAS}) (F_0 F_1...)$

2.3 Register Set

Figure 1-1 shows the 387 SX NPX register set. When an NPX is present in a system, programmers may use these registers in addition to the registers normally available on the CPU.

2.3.1 DATA REGISTERS

387 SX NPX computations use the NPX's data registers. These eight 80-bit registers provide the equivalent capacity of 20 32-bit registers. Each of the eight data registers in the NPX is 80 bits wide and is divided into "fields" corresponding to the NPX's extended-precision real data type.

The NPX register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as individually addressable registers. The TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by one and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by one. The NPX register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to use. This explicit register addressing is also relative to TOP.

2.3.2 TAG WORD

The tag word marks the content of each numeric data register, as Figure 2-1 shows. Each two-bit tag represents one of the eight data registers. The principal function of the tag word is to optimize the NPX's performance and stack handling by making it possible to distinguish between empty and nonempty register locations. It also enables exception handlers to identify special values (e.g. NaNs or denormals) in the contents of a stack location without the need to perform complex decoding of the actual data.

2.3.3 STATUS WORD

The 16-bit status word (in the status register) shown in Figure 2-2 reflects the overall state of the NPX. It may be read and inspected by programs.

Bit 15, the B-bit (busy bit) is included for 8087 compatibility only. It always has the same value as the ES bit (bit 7 of the status word); it does **not** indicate the status of the BUSY# output of NPX.

Bits 13-11 (TOP) point to the NPX register that is the current top-of-stack.

The four numeric condition code bits (C₃-C₀) are similar to the flags in a CPU; instructions that perform arithmetic operations update these bits to reflect the outcome. The effects of these instructions on the condition code are summarized in Tables 2-2 through 2-5.

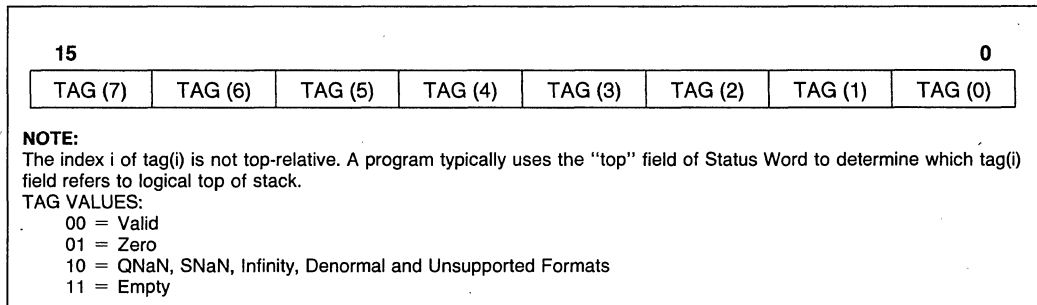


Figure 2-1. Tag Word

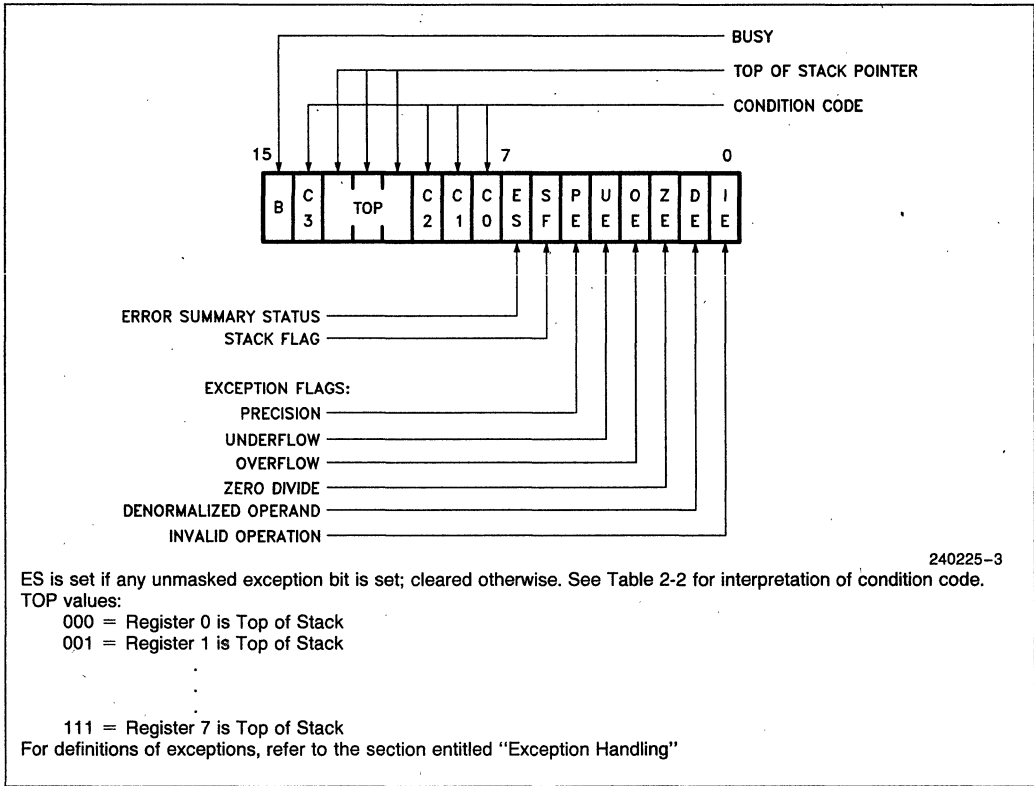


Figure 2-2. Status Word

Bit 7 is the error summary (ES) status bit. This bit is set if any unmasked exception bit is set; it is clear otherwise. If this bit is set, the ERROR# signal is asserted.

Bit 6 is the stack flag (SF). This bit is used to distinguish invalid operations due to stack overflow or underflow from other kinds of invalid operations. When SF is set, bit 9 (C₁) distinguishes between stack overflow (C₁ = 1) and underflow (C₁ = 0).

Figure 2-2 shows the six exception flags in bits 5–0 of the status word. Bits 5–0 are set to indicate that the NPX has detected an exception while executing an instruction. A later section entitled "Exception Handling" explains how they are set and used.

Note that when a new value is loaded into the status word by the FLDENV or FRSTOR instruction, the value of ES (bit 7) and its reflection in the B-bit (bit 15) are not derived from the values loaded from memory but rather are dependent upon the values of the exception flags (bits 5–0) in the status word and their corresponding masks in the control word. If ES is set in such a case, the ERROR# output of the NPX is activated immediately.

Table 2-2. Condition Code Interpretation

Instruction	C0 (S)	C3 (Z)	C1 (A)	C2 (C)
FPREM, FPREM1 (see Table 2.3)	Three least significant bits of quotient Q2 Q0		Q1 or O/U#	Reduction 0 = complete 1 = incomplete
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 2.4)		Zero or O/U#	Operand is not comparable (Table 2.4)
FXAM	Operand class (see Table 2.5)		Sign or O/U#	Operand class (Table 2.5)
FCHS, FABS, FXCH, FINCSTP, FDECSTP, Constant loads, FXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/U#	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/U#	UNDEFINED
FPTAN, FSIN FCOS, FSINCOS	UNDEFINED		Roundup or O/U#, undefined if C2 = 1	Reduction 0 = complete 1 = incomplete
FLDENV, FRSTOR	Each bit loaded from memory			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FINIT, FSAVE	UNDEFINED			
O/U#	When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).			
Reduction	If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS, and FSINCOS, the reduction bit is set if the operand at the top of the stack is too large. In this case the original operand remains at the top of the stack.			
Roundup	When the PE bit of the status word is set, this bit indicates whether the last rounding in the instruction was upward.			
UNDEFINED	Do not rely on finding any specific value in these bits.			

Table 2-3. Condition Code Interpretation after FPREM and FPREM1 Instructions

Condition Code				Interpretation after FPREM and FPREM1	
C2	C3	C1	C0		
1	X	X	X	Incomplete Reduction: further iteration required for complete reduction	
0	Q1	Q0	Q2	Q MOD8	Complete Reduction: C0, C3, C1 contain three least significant bits of quotient
	0	0	0	0	
	0	1	0	1	
	1	0	0	2	
	1	1	0	3	
	0	0	1	4	
	0	1	1	5	
	1	0	1	6	
1	1	1	7		

Table 2-4. Condition Code Resulting from Comparison

Order	C3	C2	C0
TOP > Operand	0	0	0
TOP < Operand	0	0	1
TOP = Operand	1	0	0
Unordered	1	1	1

Table 2.5. Condition Code Defining Operand Class

C3	C2	C1	C0	Value at TOP
0	0	0	0	+ Unsupported
0	0	0	1	+ NaN
0	0	1	0	Unsupported
0	0	1	1	- NaN
0	1	0	0	+ Normal
0	1	0	1	+ Infinity
0	1	1	0	- Normal
0	1	1	1	- Infinity
1	0	0	0	+ 0
1	0	0	1	+ Empty
1	0	1	0	- 0
1	0	1	1	- Empty
1	1	0	0	+ Denormal
1	1	1	0	- Denormal

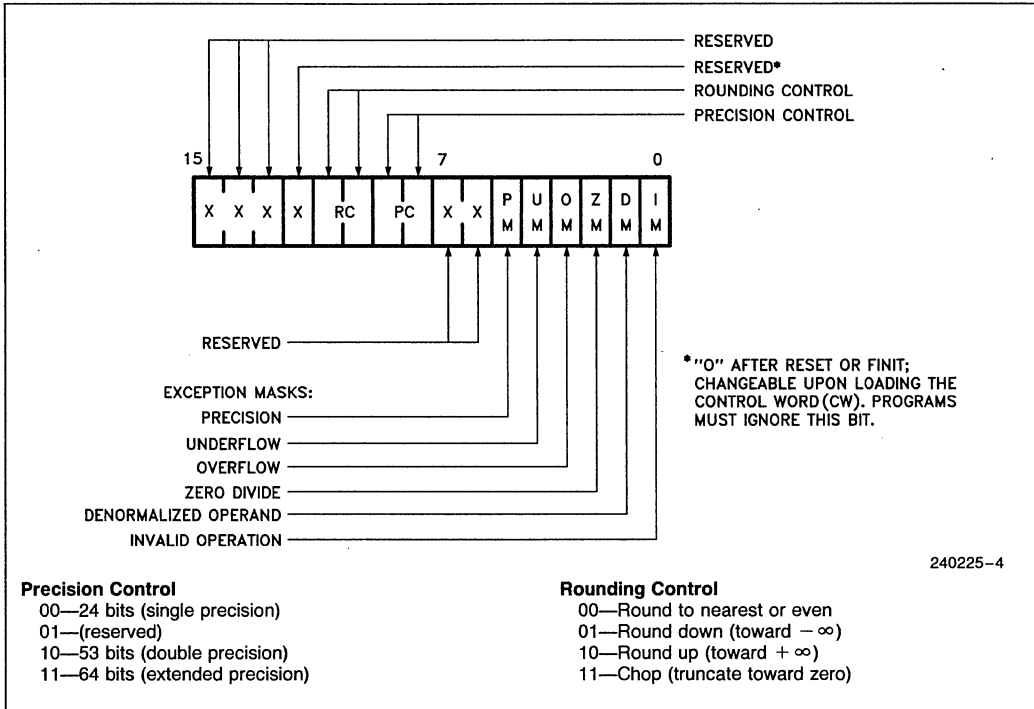


Figure 2-3. Control Word

2.3.4 CONTROL WORD

The NPX provides several processing options that are selected by loading a control word from memory into the control register. Figure 2-3 shows the format and encoding of fields in the control word.

The low-order byte of this control word configures exception masking. Bits 5–0 of the control word contain individual masks for each of the six exceptions that the NPX recognizes.

The high-order byte of the control word configures the NPX operating mode, including precision, rounding, and infinity control.

- The “infinity control bit” (bit 12) is not meaningful to the 387 SX NPX, and programs must ignore its value. To maintain compatibility with the 8087 and 80287, this bit can be programmed; however, regardless of its value, the 387 SX NPX always treats infinity in the affine sense ($-\infty < +\infty$). This bit is initialized to zero both after a hardware reset and after the FINIT instruction.

- The rounding control (RC) bits (bits 11–10) provide for directed rounding and true chop, as well as the unbiased round to nearest even mode specified in the IEEE standard. Rounding control affects only those instructions that perform rounding at the end of the operation (and thus can generate a precision exception); namely, FST, FSTP, FIST, all arithmetic instructions (except FPREM, FPREM1, FXTRACT, FABS, and FCHS), and all transcendental instructions.
- The precision control (PC) bits (bits 9–8) can be used to set the NPX internal operating precision of the significand at less than the default of 64 bits (extended precision). This can be useful in providing compatibility with early generation arithmetic processors of smaller precision. PC affects only the instructions ADD, SUB, DIV, MUL, and SQRT. For all other instructions, either the precision is determined by the opcode or extended precision is used.

2.3.5 INSTRUCTION AND DATA POINTERS

Because the NPX operates in parallel with the CPU, any exceptions detected by the NPX may be reported after the CPU has executed the ESC instruction which caused it. To allow identification of the failing numeric instruction, the 386 SX Microprocessor and 387 SX Math Coprocessor contains registers that aid in diagnosis. These registers supply the address of the failing instruction and the address of its numeric memory operand (if appropriate).

The instruction and data pointers are provided for user-written exception handlers. These registers are actually located in the CPU, but appear to be located in the NPX because they are accessed by the ESC instructions FLDENV, FSTENV, FSAVE, and

FRSTOR. Whenever the CPU executes a new ESC instruction, it saves the address of the instruction (including any prefixes that may be present), the address of the operand (if present), and the opcode.

The instruction and data pointers appear in one of four formats depending on the operating mode of the CPU (protected mode or real-address mode) and depending on the operand-size attribute in effect (32-bit operand or 16-bit operand). (See Figures 2-4, 2-5, 2-6, and 2-7.) The ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR are used to transfer these values between the registers and memory. Note that the value of the data pointer is *undefined* if the prior ESC instruction did not have a memory operand.

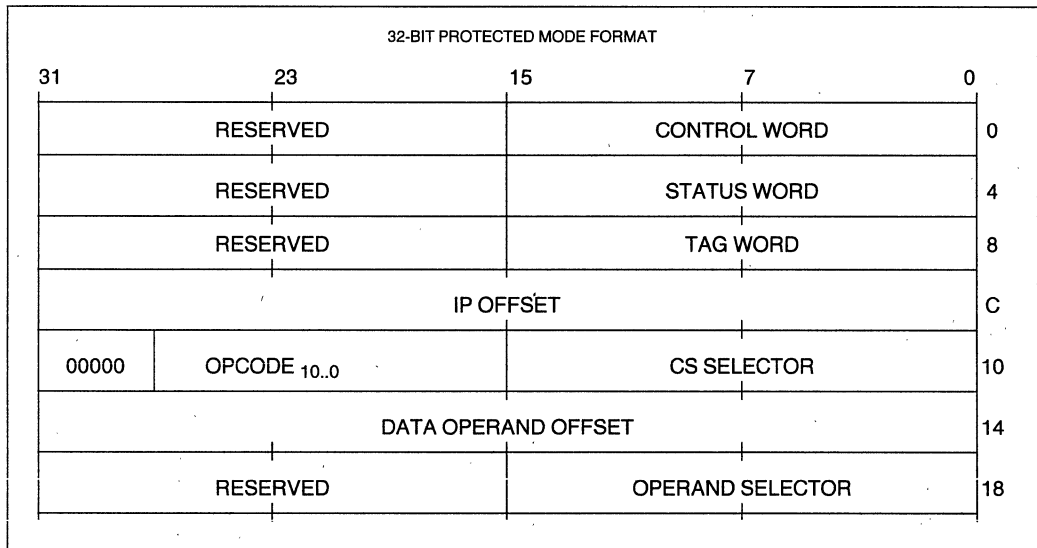


Figure 2-4. Instruction and Data Pointer Image in Memory, 32-bit Protected-Mode Format

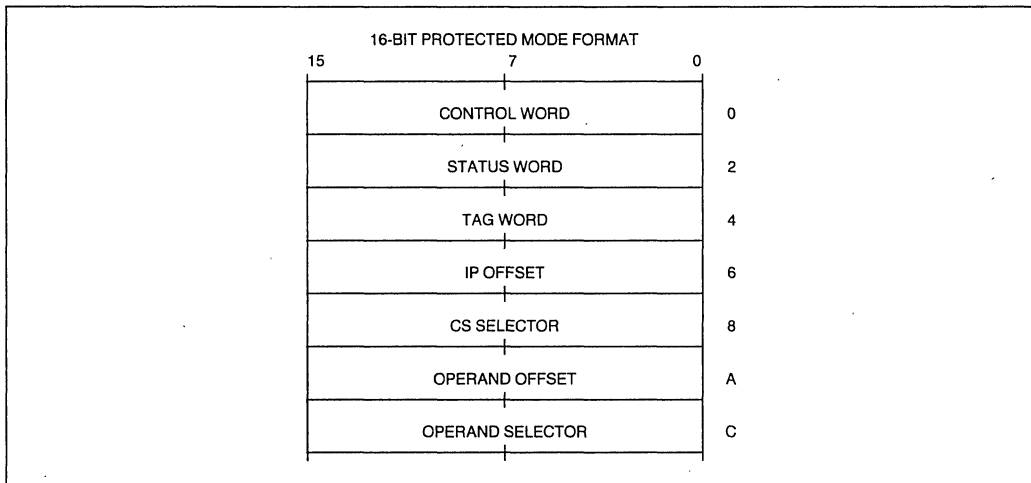


Figure 2-5. Instruction and Data Pointer Image in Memory, 16-bit Protected-Mode Format

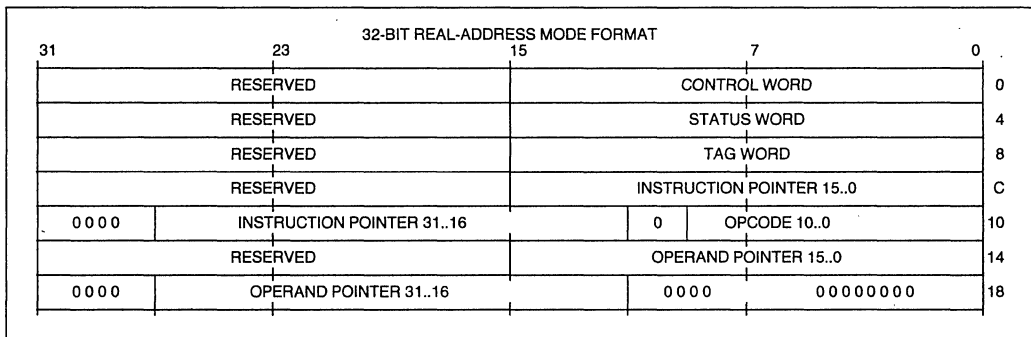


Figure 2-6. Instruction and Data Pointer Image in Memory, 32-bit Real-Mode Format

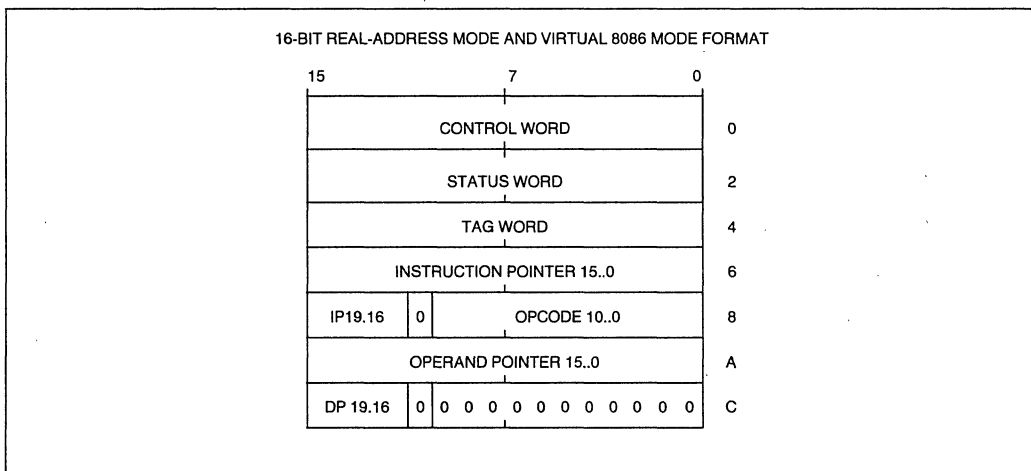


Figure 2-7. Instruction and Data Pointer Image in Memory, 16-bit Real-Mode Format

Table 2-6. CPU Interrupt Vectors Reserved for NPX

Interrupt Number	Cause of Interrupt
7	An ESC instruction was encountered when EM or TS of CPU control register zero (CR0) was set. EM = 1 indicates that software emulation of the instruction is required. When TS is set, either an ESC or WAIT instruction causes interrupt 7. This indicates that the current NPX context may not belong to the current task.
9	In a protected-mode system, an operand of a coprocessor instruction wrapped around an addressing limit (0FFFFH for expand-up segments, zero for expand-down segments) and spanned inaccessible addresses ^a . The failing numerics instruction is not restartable. The address of the failing numerics instruction and data operand may be lost; an FSTENV does not return reliable addresses. The segment overrun exception should be handled by executing an FNINIT instruction (i.e. an FINIT without a preceding WAIT). The exception can be avoided by never allowing numerics operands to cross the end of a segment.
13	In a protected-mode system, the first word of a numeric operand is not entirely within the limit of its segment. The return address pushed onto the stack of the exception handler points at the ESC instruction that caused the exception, including any prefixes. The NPX has not executed this instruction; the instruction pointer and data pointer register refer to a previous, correctly executed instruction.
16	The previous numerics instruction caused an unmasked exception. The address of the faulty instruction and the address of its operand are stored in the instruction pointer and data pointer registers. Only ESC and WAIT instructions can cause this interrupt. The CPU return address pushed onto the stack of the exception handler points to a WAIT or ESC instruction (including prefixes). This instruction can be restarted after clearing the exception condition in the NPX. FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE cannot cause this interrupt.

a. An operand may wrap around an addressing limit when the segment limit is near an addressing limit and the operand is near the largest valid address in the segment. Because of the wrap-around, the beginning and ending addresses of such an operand will be at opposite ends of the segment. There are two ways that such an operand may also span inaccessible addresses: 1) if the segment limit is not equal to the addressing limit (e.g. addressing limit is FFFFH and segment limit is FFFDH) the operand will span addresses that are not within the segment (e.g. an 8-byte operand that starts at valid offset FFFCH will span addresses FFFC–FFFFH and 0000-0003H; however addresses FFEH and FFFFH are not valid, because they exceed the limit); 2) if the operand begins and ends in present and accessible segments but intermediate bytes of the operand fall in a not-present page or in a segment or page to which the procedure does not have access rights.

2.4 Interrupt Description

CPU interrupts are used to report exceptional conditions while executing numeric programs in either real or protected mode. Table 2-6 shows these interrupts and their functions.

2.5 Exception Handling

The NPX detects six different exception conditions that can occur during instruction execution. Table 2-7 lists the exception conditions in order of precedence, showing for each the cause and the default action taken by the NPX if the exception is masked by its corresponding mask bit in the control word.

Any exception that is not masked by the control word sets the corresponding exception flag of the status word, sets the ES bit of the status word, and asserts the ERROR# signal. When the CPU attempts to execute another ESC instruction or WAIT, exception 16 occurs. The exception condition must be resolved via an interrupt service routine. The return address pushed onto the CPU stack upon entry

to the service routine does not necessarily point to the failing instruction nor to the following instruction. The CPU saves the address of the floating-point instruction that caused the exception and the address of any memory operand required by that instruction.

2.6 Initialization

After FNINIT or RESET, the control word contains the value 037FH (all exceptions masked, precision control 64 bits, rounding to nearest) the same values as in an 80287 after RESET. For compatibility with the 8087 and 80287, the bit that used to indicate infinity control (bit 12) is set to zero; however, regardless of its setting, infinity is treated in the affine sense. After FNINIT or RESET, the status word is initialized as follows:

- All exceptions are set to zero.
- Stack TOP is zero, so that after the first push the stack top will be register seven (111B).
- The condition code C₃–C₀ is undefined.
- The B-bit is zero.

Table 2-7. Exceptions

Exception	Cause	Default Action (if exception is masked)
Invalid Operation	Operation on a signalling NaN, unsupported format, indeterminate for $(0-\infty, 0/0, (+\infty) + (-\infty), \text{etc.})$, or stack overflow/underflow (SF is also set)	Result is a quiet NaN, integer indefinite, or BCD indefinite
Denormalized Operand	At least one of the operands is denormalized, i.e., it has the smallest exponent but a nonzero significand.	Normal processing continues
Zero Divisor	The divisor is zero while the dividend is a noninfinite, nonzero number	Result is ∞
Overflow	The result is too large in magnitude to fit in the specified format	Result is largest finite value or ∞
Underflow	The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes the loss of accuracy.	Result is denormalized or zero
Inexact Result (Precision)	The true result is not exactly representable in the specified format (e.g. $1/3$); the result is rounded according to the rounding mode.	Normal processing continues

The tag word contains FFFFH (all stack locations are empty).

The 386 SX Microprocessor and 387 SX Math Coprocessor initialization software must execute an FNINIT instruction (i.e. an FINIT without a preceding WAIT) after RESET. The FNINIT is not strictly required for the 80287 software, but Intel recommends its use to help ensure upward compatibility with other processors. After a hardware RESET, the ER-ROR# output is asserted to indicate that a 387 SX NPX is present. To accomplish this, the IE and ES bits of the status word are set, and the IM bit in the control word is cleared. After FNINIT, the status word and the control word have the same values as in an 80287 after RESET.

2.7 8087 and 80287 Compatibility

This section summarizes the differences between the 387 SX NPX and the 80287. Any migration from the 8087 directly to the 387 SX NPX must also take into account the differences between the 8087 and the 80287 as listed in Appendix A.

Many changes have been designed into the 387 SX NPX to directly support the IEEE standard in hardware. These changes result in increased performance by eliminating the need for software that supports the standard.

2.7.1 GENERAL DIFFERENCES

The 387 SX NPX supports only affine closure for infinity arithmetic, not projective closure.

Operands for FSCALE and FPATAN are no longer restricted in range (except for $\pm\infty$); F2XM1 and FPTAN accept a wider range of operands.

Rounding control is in effect for FLD *constant*.

Software cannot change entries of the tag word to values (other than empty) that differ from actual register contents.

After reset, FINIT, and incomplete FPREM, the 387 SX NPX resets to zero the condition code bits C_3-C_0 of the status word.

In conformance with the IEEE standard, the 387 SX NPX does not support the special data formats pseudozero, pseudo-NaN, pseudoinfinity, and unnormal.

The denormal exception has a different purpose on the 387 SX NPX. A system that uses the denormal-exception handler solely to normalize the denormal operands, would better mask the denormal exception on the 387 SX NPX. The 387 SX NPX automatically normalizes denormal operands when the denormal exception is masked.

2.7.2 EXCEPTIONS

A number of differences exist due to changes in the IEEE standard and to functional improvements to the architecture of the 387 SX NPX:

1. When the overflow or underflow exception is masked, the 387 SX NPX differs from the 80287 in rounding when overflow or underflow occurs. The 387 SX NPX produces results that are consistent with the rounding mode.
2. When the underflow exception is masked, the 387 SX NPX sets its underflow flag only if there is also a loss of accuracy during denormalization.
3. Fewer invalid-operation exceptions due to denormal operands, because the instructions FSQRT, FDIV, FPREM, and conversions to BCD or to integer normalize denormal operands before proceeding.
4. The FSQRT, FBSTP, and FPREM instructions may cause underflow, because they support denormal operands.
5. The denormal exception can occur during the transcendental instructions and the FXTRACT instruction.
6. The denormal exception no longer takes precedence over all other exceptions.
7. When the denormal exception is masked, the 387 SX NPX automatically normalizes denormal operands. The 8087/80287 performs unnormal arithmetic, which might produce an unnormal result.
8. When the operand is zero, the FXTRACT instruction reports a zero-divide exception and leaves $-\infty$ in ST(1).
9. The status word has a new bit (SF) that signals when invalid-operation exceptions are due to stack underflow or overflow.
10. FLD *extended precision* no longer reports denormal exceptions, because the instruction is not numeric.
11. FLD *single/double precision* when the operand is denormal converts the number to extended precision and signals the denormalized operand exception. When loading a signalling NaN, FLD *single/double precision* signals an invalid-operand exception.
12. The 387 SX NPX only generates quiet NaNs (as on the 80287); however, the 387 SX NPX distinguishes between quiet NaNs and signaling NaNs. Signaling NaNs trigger exceptions when they are used as operands; quiet NaNs do not (except for FCOM, FIST, and FBSTP which also raise IE for quiet NaNs).
13. When stack overflow occurs during FPTAN and overflow is masked, both ST(0) and ST(1) con-

tain quiet NaNs. The 80287/8087 leaves the original operand in ST(1) intact.

14. When the scaling factor is $\pm\infty$, the FSCALE (ST(0), ST(1)) instruction behaves as follows (ST(0) and ST(1) contain the scaled and scaling operands respectively):
 - FSCALE(0, ∞) generates the invalid operation exception.
 - FSCALE(finite, $-\infty$) generates zero with the same sign as the scaled operand.
 - FSCALE(finite, $+\infty$) generates ∞ with the same sign as the scaled operand.

The 8087/80287 returns zero in the first case and raises the invalid-operation exception in the other cases.
15. The 387 SX NPX returns signed infinity/zero as the unmasked response to massive overflow/underflow. The 8087 and 80287 support a limited range for the scaling factor; within this range either massive overflow/underflow do not occur or undefined results are produced.

3.0 HARDWARE INTERFACE

In the following description of hardware interface, the # symbol at the end of a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no # is present after the signal name, the signal is asserted when at the high voltage level.

3.1 Signal Description

In the following signal descriptions, the 387 SX NPX pins are grouped by function as shown by Table 3-1. Table 3-1 lists every pin by its identifier, gives a brief description of its function, and lists some of its characteristics (Refer to Figure 5-1 and Table 5-1 for pin configuration).

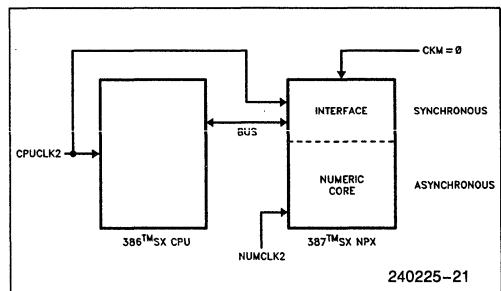


Figure 3.1. Asynchronous Operation

Table 3-1. Pin Summary

Pin Name	Function	Active State	Input/Output	Referenced To...
Execution Control				
CPUCLK2	386™ SX Microprocessor CLoCK 2			
NUMCLK2	NPX CLoCK 2			
CKM	NPX CloCkIng Mode			
RESETIN	System reset	High		CPUCLK2
NPX Handshake				
PEREQ	Processor Extension REQuest	High	O	STEN/CPUCLK2
BUSY #	Busy status	Low	O	STEN/CPUCLK2
ERROR #	Error status	Low	O	STEN/NUMCLK2
Bus Interface				
D15–D0	Data pins	High	I/O	CPUCLK2
W/R #	Write/Read bus cycle	Hi/Lo		CPUCLK2
ADS #	ADdress Strobe	Low		CPUCLK2
READY #	Bus ready input	Low		CPUCLK2
READYO #	Ready output	Low	O	STEN/CPUCLK2
Chip/Port Select				
STEN	STatus ENable	High		CPUCLK2
NPS1 #	NPX select # 1	Low		CPUCLK2
NPS2	NPX select # 2	High		CPUCLK2
CMD0 #	CoMmanD	Low		CPUCLK2
Power and Ground				
V _{CC}	System power			
V _{SS}	System ground			

All output signals are tristate; they leave floating state only when STEN is active. The output buffers of the bidirectional data pins D15–D0 are also tristate; they leave floating state only during cycles when the NPX is selected (i.e. when STEN, NPS1 #, and NPS2 are all active).

3.1.1 386™ SX CPU CLOCK 2 (CPUCLK2)

This input uses the CLK2 signal of the CPU to time the bus control logic. Several other NPX signals are referenced to the rising edge of this signal. When CKM = 1 (synchronous mode) this pin also clocks the data interface and control unit and the floating-point unit of the NPX. This pin requires MOS-level input. The signal on this pin is divided by two to produce the internal clock signal CLK.

3.1.2 387™ SX NPX CLOCK 2 (NUMCLK2)

When CKM = 0 (asynchronous mode) this pin provides the clock for the data interface and control unit and the floating-point unit of the NPX. In this case, the ratio of the frequency of NUMCLK2 to the frequency of CPUCLK2 must lie within the range 10:16 to 14:10. When CKM = 1 (synchronous mode) signals on this pin are ignored; CPUCLK2 is used instead for the data interface and control unit and the floating-point unit. This pin requires MOS-level input.

3.1.3 CLOCKING MODE (CKM)

This pin is a strapping option. When it is strapped to V_{CC} (HIGH), the NPX operates in synchronous mode; when strapped to V_{SS} (LOW), the NPX operates in asynchronous mode. These modes relate to clocking of the data interface and control unit and the floating-point unit only; the bus control logic always operates synchronously with respect to the CPU.

3.1.4 SYSTEM RESET (RESETIN)

A LOW to HIGH transition on this pin causes the NPX to terminate its present activity and to enter a dormant state. RESETIN must remain active (HIGH) for at least 40 NUMCLK2 periods.

The HIGH to LOW transitions of RESETIN must be synchronous with CPUCLK2, so that the phase of the internal clock of the bus control logic (which is the CPUCLK2 divided by two) is the same as the phase of the internal clock of the CPU. After RESETIN goes LOW, at least 50 NUMCLK2 periods must pass before the first NPX instruction is written into the NPX. This pin should be connected to the CPU RESET pin. Table 3-1 shows the status of the output pins during the reset sequence. After a reset, all output pins return to their inactive states.

Table 3-2. Output Pin Status during Reset

Pin Value	Pin Name
HIGH	READYO#, BUSY#
LOW	PEREQ, ERROR#
Tri-State OFF	D15-D0

3.1.5 PROCESSOR EXTENSION REQUEST (PEREQ)

When active, this pin signals to the CPU that the NPX is ready for data transfer to/from its data FIFO. When all data is written to or read from the data FIFO, PEREQ is deactivated. This signal always goes inactive before BUSY# goes inactive. This signal is referenced to CPUCLK2. It should be connected to the CPU PEREQ input.

3.1.6 BUSY STATUS (BUSY#)

When active, this pin signals to the CPU that the NPX is currently executing an instruction. This signal is referenced to CPUCLK2. It should be connected to the CPU BUSY# pin.

3.1.7 ERROR STATUS (ERROR#)

This pin reflects the ES bit of the status register. When active, it indicates that an unmasked exception has occurred. This signal can be changed to inactive state only by the following instructions (without a preceding WAIT): FNINIT, FNCLEX, FNSTENV, FNSAVE, FLDW, FLDENV, and FRSTOR. This pin is referenced to CPUCLK2. It should be connected to the ERROR# pin of the CPU.

3.1.8 DATA PINS (D15-D0)

These bidirectional pins are used to transfer data and opcodes between the CPU and NPX. They are normally connected directly to the corresponding CPU data pins. HIGH state indicates a value of one. D0 is the least significant data bit. Timings are referenced to CPUCLK2.

3.1.9 WRITE/READ BUS CYCLE (W/R#)

This signal indicates to the NPX whether the CPU bus cycle in progress is a read or a write cycle. This pin should be connected directly to the CPU's W/R# pin. HIGH indicates a write cycle; LOW a read cycle. This input is ignored if any of the signals STEN, NPS1#, or NPS2 is inactive. Setup and hold times are referenced to CPUCLK2.

3.1.10 ADDRESS STROBE (ADS#)

This input, in conjunction with the READY# input, indicates when the NPX bus-control logic may sample W/R# and the chip-select signals. Setup and hold times are referenced to CPUCLK2. This pin should be connected to the ADS# pin of the CPU.

3.1.11 BUS READY INPUT (READY#)

This input indicates to the NPX when a CPU bus cycle is to be terminated. It is used by the bus-control logic to trace bus activities. Bus cycles can be extended indefinitely until terminated by READY#. This input should be connected to the same signal that drives the CPU's READY# input. Setup and hold times are referenced to CPUCLK2.

3.1.12 READY OUTPUT (READYO#)

This pin is activated at such a time that write cycles are terminated after two clocks (except FLDENV and FRSTOR) and read cycles after three clocks. In configurations where no extra wait states are required, this pin must directly or indirectly drive the READY# input of the CPU. Refer to the section entitled "Bus Operation" for details. This pin is activated only during bus cycles that select the NPX. This signal is referenced to CPUCLK2.

3.1.13 STATUS ENABLE (STEN)

This pin serves as a chip select for the NPX. When inactive, this pin forces, BUSY#, PEREQ#, ERROR#, and READYO# outputs into floating state. D15-D0 are normally floating; they leave floating state only if STEN is active and additional conditions are met. STEN also causes the chip to recognize its other chip-select inputs. STEN makes it easier to do on-board testing (using the overdrive method) of other chips in systems containing the NPX. STEN should be pulled up with a resistor so that it can be pulled down when testing. In boards that do not use on-board testing, STEN should be connected to V_{CC}. Setup and hold times are relative to CPUCLK2. Note that STEN must maintain the same setup and hold times as NPS1#, NPS2, and CMD0# (i.e. if STEN changes state during an NPX bus cycle, it must change state during the same CLK period as the NPS1#, NPS2, and CMD0# signals).

3.1.14 NPX SELECT 1 (NPS1#)

When active (along with STEN and NPS2) in the first period of a CPU bus cycle, this signal indicates that the purpose of the bus cycle is to communicate with the NPX. This pin should be connected directly to the M/IO# pin of the CPU, so that the NPX is selected only when the CPU performs I/O cycles. Setup and hold times are referenced to CPUCLK2.

3.1.15 NPX SELECT 2 (NPS2)

When active (along with STEN and NPS1#) in the first period of a CPU bus cycle, this signal indicates that the purpose of the bus cycle is to communicate with the NPX. This pin should be connected directly to the A23 pin of the CPU, so that the NPX is selected only when the CPU issues one of the I/O addresses reserved for the NPX (8000F8H, 8000FCH or 8000FEH which is treated as 8000FCH by the NPX). Setup and hold times are referenced to CPUCLK2.

3.1.16 COMMAND (CMD0#)

During a write cycle, this signal indicates whether an opcode (CMD0# active) or data (CMD0# inactive) is being sent to the NPX. During a read cycle, it indicates whether the control or status register (CMD0# active) or a data register (CMD0# inactive) is being read. CMD0# should be connected directly to the A2 output of the CPU. Setup and hold times are referenced to CPUCLK2.

3.1.17 SYSTEM POWER (V_{CC})

System power provides the +5V DC supply input. All V_{CC} pins should be tied together on the circuit board and local decoupling capacitors should be used between V_{CC} and V_{SS}.

3.1.18 SYSTEM GROUND (V_{SS})

All V_{SS} pins should be tied together on the circuit board and local decoupling capacitors should be used between V_{CC} and V_{SS}.

3.2 System Configuration

The 387 SX Math Coprocessor is designed to interface with the 386 SX Microprocessor as shown by Figure 3-1. A dedicated communication protocol makes possible high-speed transfer of opcodes and operands between the CPU and NPX. The 387 SX NPX is designed so that no additional components are required for interface with the CPU. Most control pins of the NPX are connected directly to pins of the CPU.

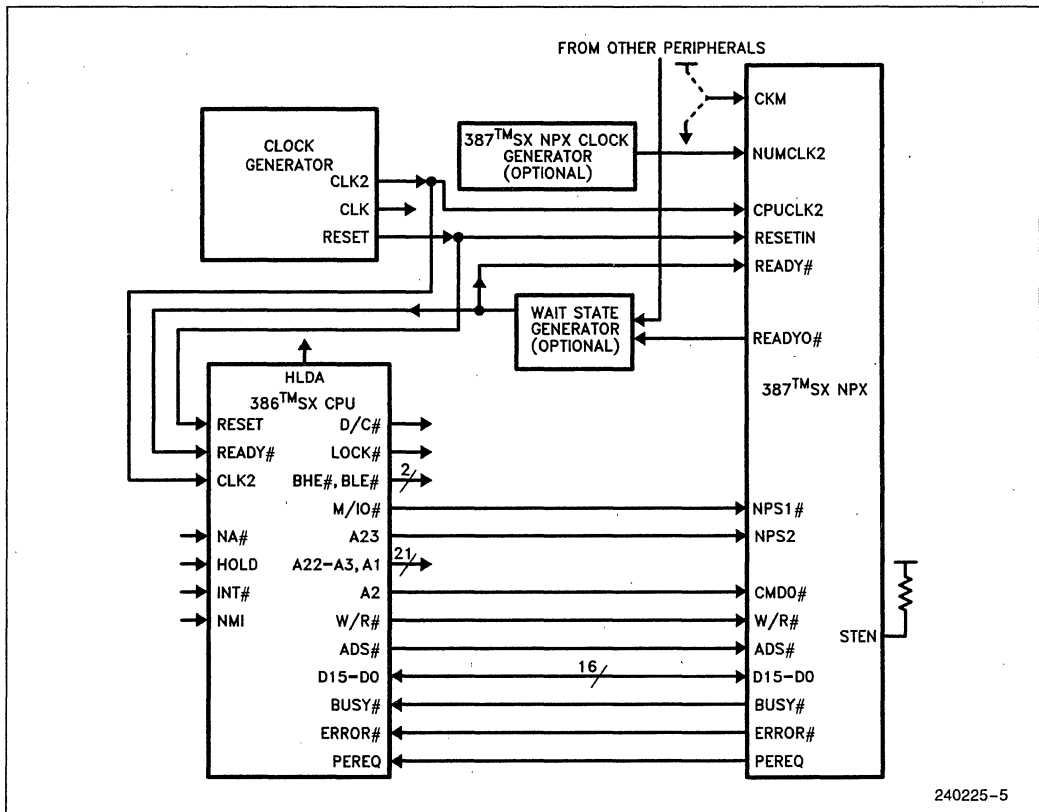


Figure 3-1. 386™ SX CPU and 387™ SX NPX System Configuration

The interface between the NPX and the CPU has these characteristics:

- The NPX shares the local bus of the 386 SX Microprocessor.
- The CPU and NPX share the same reset signals. They may also share the same clock input; however, for greatest performance, an external oscillator may be needed.
- The corresponding BUSY#, ERROR#, and PEREQ pins are connected together.
- The NPX NPS1# and NPS2 inputs are connected to the latched CPU M/I/O# and A23 outputs respectively. For coprocessor cycles, M/I/O# is always LOW and A23 always HIGH.
- The NPX input CMD0 is connected to the latched A₂ output. The 386 SX Microprocessor generates address 8000F8H when writing a command and address 8000FCH or 8000FEH (treated as 8000FCH by the 387 SX NPX) when writing or reading data. It does not generate any other addresses during NPX bus cycles.

3.3 Processor Architecture

As shown by the block diagram on the front page, the 387 SX NPX is internally divided into three sections: the bus control logic (BCL), the data interface and control unit, and the floating point unit (FPU). The FPU (with the support of the control unit which contains the sequencer and other support units) executes all numeric instructions. The data interface and control unit is responsible for the data flow to and from the FPU and the control registers, for receiving the instructions, decoding them, and sequencing the microinstructions, and for handling some of the administrative instructions. The BCL is responsible for CPU bus tracking and interface. The BCL is the only unit in the NPX that must run synchronously with the CPU; the rest of the NPX can run asynchronously with respect to the CPU.

3.3.1 BUS CONTROL LOGIC

The BCL communicates solely with the CPU using I/O bus cycles. The BCL appears to the CPU as a special peripheral device. It is special in two re-

spects: the CPU initiates I/O automatically when it encounters ESC instructions, and the CPU uses reserved I/O addresses to communicate with the BCL. The BCL does not communicate directly with memory. The CPU performs all memory access, transferring input operands from memory to the NPX and transferring outputs from the NPX to memory.

3.3.2 DATA INTERFACE AND CONTROL UNIT

The data interface and control unit latches the data and, subject to BCL control, directs the data to the FIFO or the instruction decoder. The instruction decoder decodes the ESC instructions sent to it by the CPU and generates controls that direct the data flow in the FIFO. It also triggers the microinstruction sequencer that controls execution of each instruction. If the ESC instruction is FINIT, FCLEX, FSTSW, FSTSW AX, FSTCW, FSETPM, or FRSTPM, the control executes it independently of the FPU and the sequencer. The data interface and control unit is the one that generates the BUSY#, PEREQ, and ERROR# signals that synchronize NPX activities with the CPU.

3.3.3 FLOATING-POINT UNIT

The FPU executes all instructions that involve the register stack, including arithmetic, logical, transcendental, constant, and data transfer instructions. The data path in the FPU is 84 bits wide (68 significant bits, 15 exponent bits, and a sign bit) which allows internal operand transfers to be performed at very high speeds.

3.4 Bus Cycles

The pins STEN, NPS1#, NPS2, CMD0, and W/R# identify bus cycles for the NPX. Table 3-3 defines the types of NPX bus cycles.

3.4.1 387™ SX NPX ADDRESSING

The NPS1#, NPS2, and CMD0 signals allow the NPX to identify which bus cycles are intended for the NPX. The NPX responds to I/O cycles when the I/O address is 8000F8H, 8000FCH or 8000FEH (treated

Table 3-3. Bus Cycle Definition

STEN	NPS1#	NPS2	CMD0#	W/R#	Bus Cycle Type
0	x	x	x	x	NPX not selected and all outputs in floating state
1	1	x	x	x	NPX not selected
1	x	0	x	x	NPX not selected
1	0	1	0	0	CW or SW read from NPX
1	0	1	0	1	Opcode write to NPX
1	0	1	1	0	Data read from NPX
1	0	1	1	1	Data write to NPX

as 8000FCH by the 387 SX NPX). The NPX responds to I/O cycles when bit 23 of the I/O address is set. In other words, the NPX acts as an I/O device in a reserved I/O address space.

Because A23 is used to select the 387 SX Numerics Coprocessor Extension for data transfers, it is not possible for a program running on the CPU to address the NPX with an I/O instruction. Only ESC instructions cause the CPU to communicate with the NPX.

3.4.2 CPU/NPX SYNCHRONIZATION

The pins **BUSY#**, **PEREQ**, and **ERROR#** are used for various aspects of synchronization between the CPU and the NPX.

BUSY# is used to synchronize instruction transfer from the CPU to the NPX. When the NPX recognizes an ESC instruction, it asserts **BUSY#**. For most ESC instructions, the CPU waits for the NPX to deassert **BUSY#** before sending the new opcode.

The NPX uses the **PEREQ** pin of the CPU to signal that the NPX is ready for data transfer to or from its data FIFO. The NPX does not directly access memory; rather, the CPU provides memory access services for the NPX. (For this reason, memory access on behalf of the NPX always obeys the protection rules applicable to the current CPU mode.) Once the CPU initiates an NPX instruction that has operands, the CPU waits for **PEREQ** signals that indicate when the NPX is ready for operand transfer. Once all operands have been transferred (or if the instruction has no operands) the CPU continues program execution while the NPX executes the ESC instruction.

In 8086/8087 systems, **WAIT** instructions may be required to achieve synchronization of both commands and operands. In the 386 SX Microprocessor and 387 SX Math Coprocessor systems, however, **WAIT** instructions are required only for operand synchronization; namely, after NPX stores to memory (except **FSTSW** and **FSTCW**) or load from memory. (In 80286/80287 systems, **WAIT** is required before **FLDENV** and **FRSTOR**; with the 386 SX Microprocessor and 387 SX Math Coprocessor, **WAIT** is not required in these cases.) Used this way, **WAIT** ensures that the value has already been written or read by the NPX before the CPU reads or changes the value.

Once it has started to execute a numerics instruction and has transferred the operands from the CPU, the NPX can process the instruction in parallel with and independent of the host CPU. When the NPX detects an exception, it asserts the **ERROR#** signal, which causes a CPU interrupt.

3.4.3 SYNCHRONOUS OR ASYNCHRONOUS MODES

The internal logic of the NPX (the FPU) can operate either directly from the CPU clock (synchronous mode) or from a separate clock (asynchronous mode). The two configurations are distinguished by the **CKM** pin. In either case, the bus control logic (**BCL**) of the NPX is synchronized with the CPU clock. Use of asynchronous mode allows the CPU and the FPU section of the NPX to run at different speeds. In this case, the ratio of the frequency of **NUMCLK2** to the frequency of **CPUCLK2** must lie within the range 10:16 to 14:10. Use of synchronous mode eliminates one clock generator from the board design.

3.4.4 AUTOMATIC BUS CYCLE TERMINATION

In configurations where no extra wait states are required, **READYO#** can drive the CPU's **READY#** input. If this pin is used, it should be connected to the logic that ORs all **READY** outputs from peripherals on the CPU bus. **READYO#** is asserted by the NPX only during I/O cycles that select the NPX. Refer to Section 4.0 "Bus Operation" for details.

4.0 BUS OPERATION

With respect to bus interface, the 387 SX NPX is fully synchronous with the CPU. Both operate at the same rate, because each generates its internal **CLK** signal by dividing **CPUCLK2** by two. Furthermore, both internal **CLK** signals are in phase, because they are synchronized by the same **RESETIN** signal.

A bus cycle for the NPX starts when the CPU activates **ADS#** and drives new values on the address and cycle-definition lines. The NPX examines the address and cycle-definition lines in the same **CLK** period during which **ADS#** is activated. This **CLK** period is considered the first **CLK** of the bus cycle. During this first **CLK** period, the NPX also examines the **R/W#** input signal to determine whether the cycle is a read or a write cycle and examines the **CMD0** input to determine whether an opcode, operand, or control/status register transfer is to occur.

The 387 SX NPX supports both pipelined (i.e. overlapped) and nonpipelined bus cycles. A nonpipelined cycle is one for which the CPU asserts **ADS#** when no other NPX bus cycle is in progress. A pipelined bus cycle is one for which the CPU asserts **ADS#** and provides valid next-address and control signals before the prior NPX cycle terminates. The CPU may do this as early as the second **CLK** period after asserting **ADS#** for the prior cycle. Pipelining increas-

es the availability of the bus by at least one CLK period. The 387 SX NPX supports pipelined bus cycles in order to optimize address pipelining by the CPU for memory cycles.

Bus operation is described in terms of an abstract state machine. Figure 4-1 illustrates the states and state transitions for NPX bus cycles:

- T_I is the idle state. This is the state of the bus logic after RESET, the state to which bus logic returns after every nonpipelined bus cycle, and the state to which bus logic returns after a series of pipelined cycles.
- T_{RS} is the READY#-sensitive state. Different types of bus cycles may require a minimum of one or two successive T_{RS} states. The bus logic remains in T_{RS} state until READY# is sensed, at which point the bus cycle terminates. Any number of wait states may be implemented by delaying READY#, thereby causing additional successive T_{RS} states.
- T_P is the first state for every pipelined bus cycle. This state is not used by nonpipelined cycles.

Note that the bus logic tracks bus state regardless of the values on the chip/port select pins.

The READYO# output of the NPX indicates when an NPX bus cycle may be terminated if no extra wait states are required. For all write cycles (except those for the instructions FLDENV and FRSTOR), READYO# is always asserted during the first T_{RS} state, regardless of the number of wait states. For all read cycles and write cycles for FLDENV and

FRSTOR, READYO# is always asserted in the second T_{RS} state, regardless of the number of wait states. These rules apply to both pipelined and non-pipelined cycles. Systems designers may use READYO# in one of the following ways:

1. Connect it (directly or through logic that ORs READY# signals from other devices) to the READY# inputs of the CPU and NPX.
2. Use it as one input to a wait-state generator.

The following sections illustrate different types of 387 SX NPX bus cycles. Because different instructions have different amounts of overhead before, between, and after operand transfer cycles, it is not possible to represent in a few diagrams all of the combinations of successive operand transfer cycles. The following bus-cycle diagrams show memory cycles between NPX operand-transfer cycles. Note however that, during FRSTOR, some consecutive accesses to the NPX do not have intervening memory accesses. For the timing relationship between operand transfer cycles and opcode write or other overhead activities, see the figure "Other Parameters" in section 6.

4.1 Nonpipelined Bus Cycles

Figure 4-2 illustrates bus activity for consecutive nonpipelined bus cycles.

At the second clock of the bus cycle, the NPX enters the T_{RS} state. During this state, it samples the READY# input and stays in this state as long as READY# is inactive.

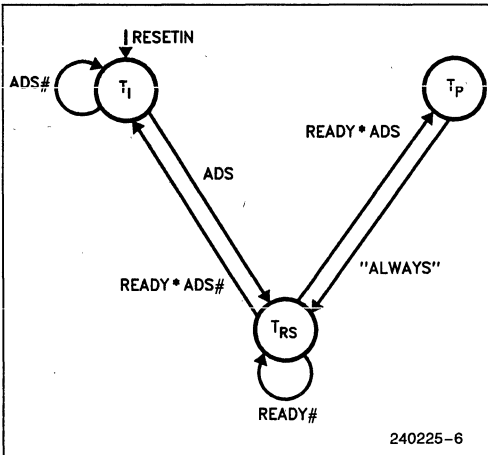


Figure 4-1. Bus State Diagram

4.1.1 WRITE CYCLE

In write cycles, the NPX drives the READYO# signal for one CLK period during the second CLK period of the cycle (i.e. the first T_{RS} state); therefore, the fastest write cycle takes two CLK periods (see cycle 2 of Figure 4-2). For the instructions FLDENV and FRSTOR, however, the NPX forces a wait state by delaying the activation of READYO# to the second T_{RS} state (not shown in Figure 4-2).

The NPX samples the D15-D0 inputs into data latches at the falling edge of CLK as long as it stays in T_{RS} state.

When READY# is asserted, the NPX returns to the idle state. Simultaneously with the NPX's entering the idle state, the CPU may assert ADS# again, signaling the beginning of yet another cycle.

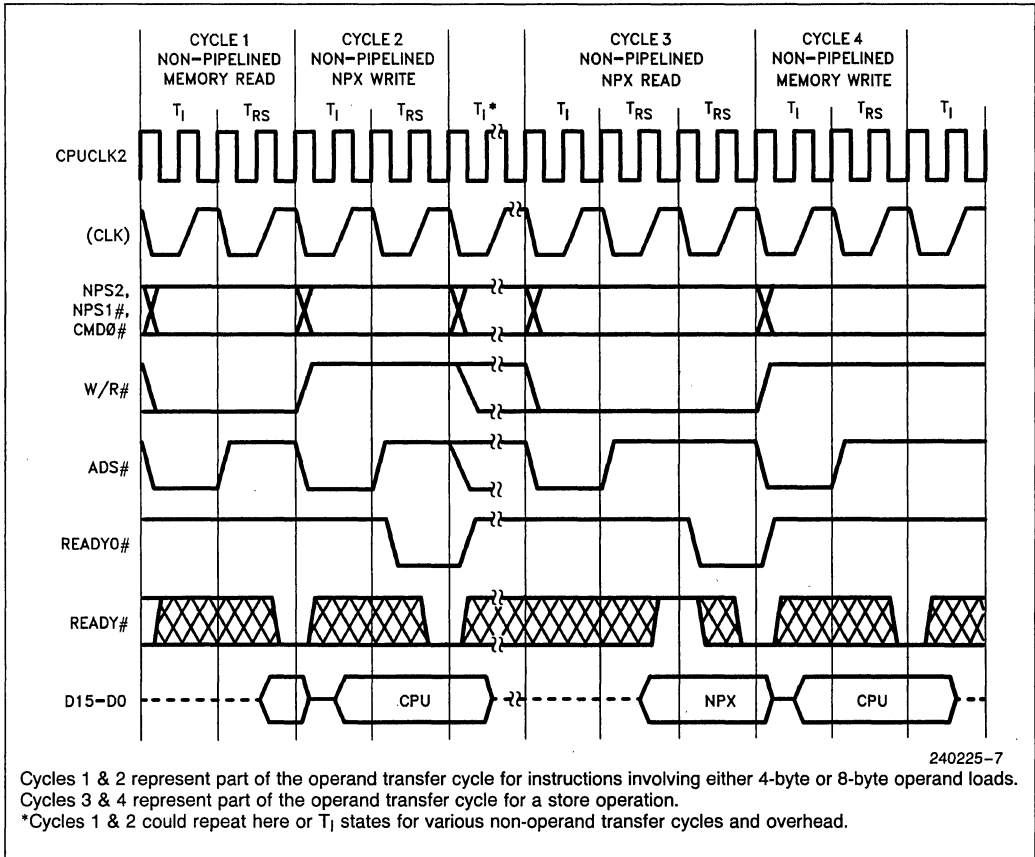


Figure 4-2. Nonpipelined Read and Write Cycles

4.1.2 READ CYCLE

At the rising edge of CLK in the second CLK period of the cycle (i.e. the first T_{RS} state), the NPX starts to drive the D15-D0 outputs and continues to drive them as long as it stays in T_{RS} state.

At least one wait state must be inserted to ensure that the CPU latches the correct data. Because the NPX starts driving the data bus only at the rising edge of CLK in the second clock period of the bus cycle, not enough time is left for the data signals to propagate and be latched by the CPU before the next falling edge of CLK. Therefore, the NPX does not drive the $READY_0\#$ signal until the third CLK period of the cycle. Thus, if the $READY_0\#$ output drives the CPU's $READY\#$ input, one wait state is automatically inserted.

Because one wait state is required for NPX reads, the minimum length of an NPX read cycle is three CLK periods, as cycle 3 of Figure 4-2 shows.

When $READY\#$ is asserted, the NPX returns to the idle state. Simultaneously with the NPX's entering the idle state, the CPU may assert $ADS\#$ again, signaling the beginning of yet another cycle. The transition from T_{RS} state to idle state causes the NPX to put the tristate D15-D0 outputs into the floating state, allowing another device to drive the data bus.

4.2 Pipelined Bus Cycles

Because all the activities of the NPX bus interface occur either during the T_{RS} state or during the transitions to or from that state, the only difference between a pipelined and a nonpipelined cycle is the manner of changing from one state to another. The exact activities during each state are detailed in the previous section "Nonpipelined Bus Cycles".

When the CPU asserts $ADS\#$ before the end of a bus cycle, both $ADS\#$ and $READY\#$ are active dur-

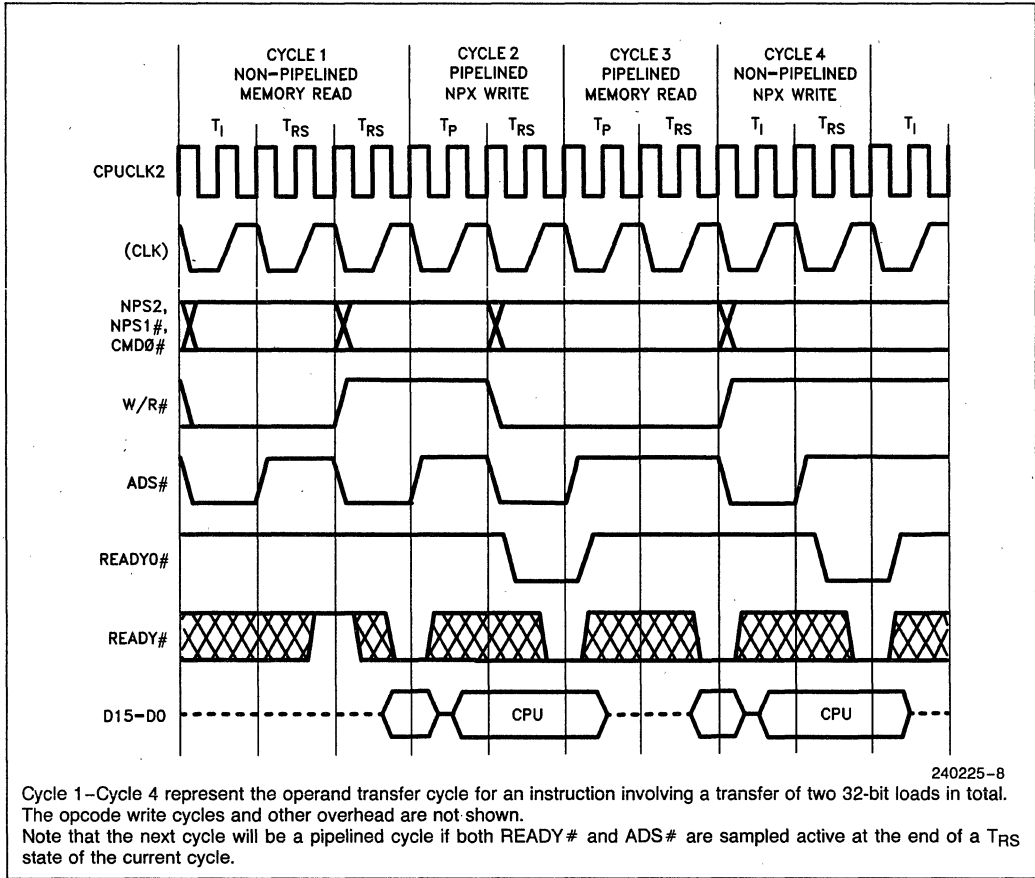


Figure 4-3. Fastest Transitions to and from Pipelined Cycles

ing a T_{RS} state. This condition causes the NPX to change to a different state named T_P. One clock period after a T_P state, the NPX always returns to T_{RS} state. In consecutive pipelined cycles, the NPX bus logic uses only the T_{RS} and T_P states.

Figure 4-3 shows the fastest transitions into and out of the pipelined bus cycles. Cycle 1 in the figure represents a nonpipelined cycle. (Nonpipelined write cycles with only one T_{RS} state (i.e. no wait states) are always followed by another nonpipelined cycle, because READY# is asserted before the earliest possible assertion of ADS# for the next cycle.)

Figure 4-4 shows pipelined write and read cycles with one additional T_{RS} state beyond the minimum required. To delay the assertion of READY# requires external logic.

4.3 Bus Cycles of Mixed Type

When the NPX bus logic is in the T_{RS} state, it distinguishes between nonpipelined and pipelined cycles according to the behavior of ADS# and READY#. In a nonpipelined cycle, only READY# is activated, and the transition is from T_{RS} state to idle state. In a

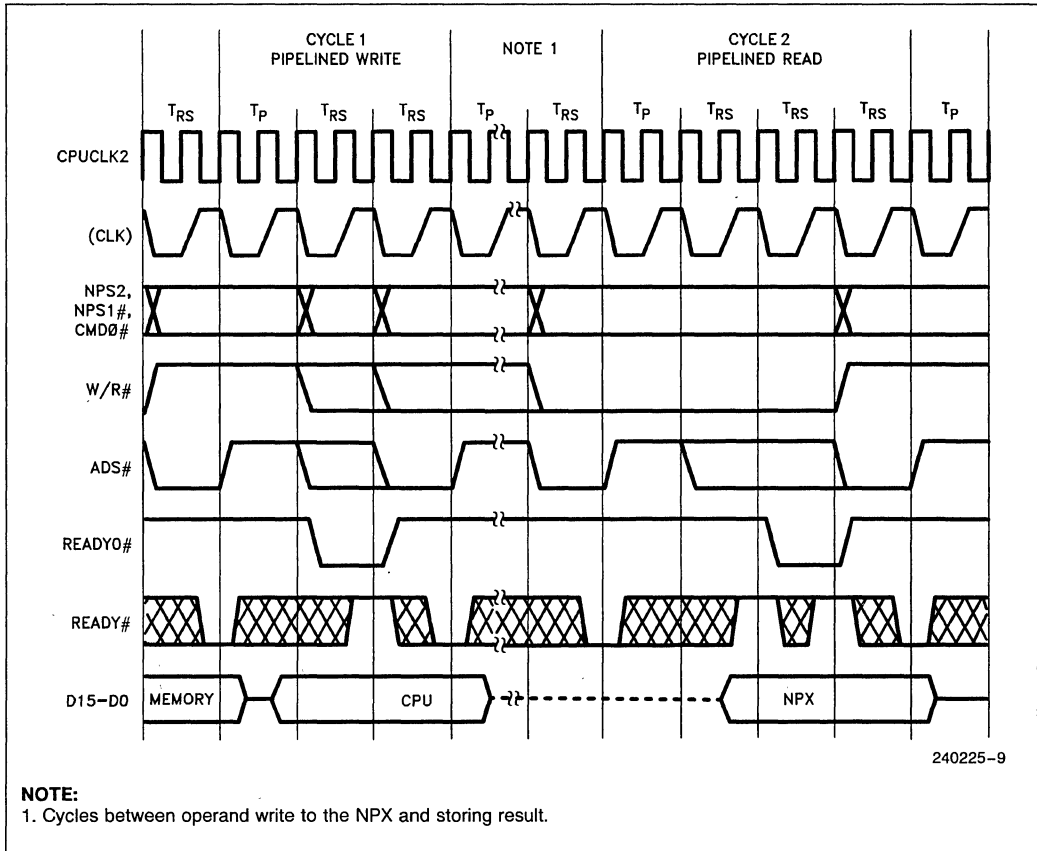


Figure 4-4. Pipelined Cycles with Wait States

pipelined cycle, both $READY\#$ and $ADS\#$ are active, and the transition is first from T_{RS} state to T_P state, then, after one clock period, back to T_{RS} state.

4.4 $BUSY\#$ and $PEREQ$ Timing Relationship

Figure 4-5 shows the activation of $BUSY\#$ at the beginning of instruction execution and its deactiva-

tion upon completion of the instruction. $PEREQ$ is activated within this interval. If $ERROR\#$ (not shown in the figure) is ever asserted, it would be asserted at least six $CPUCLK2$ periods after the deactivation of $PEREQ$ and would be deasserted at least six $CPUCLK2$ periods before the deactivation of $BUSY\#$. Figure 4-5 also shows that $STEN$ is activated at the beginning of an NPX bus cycle.

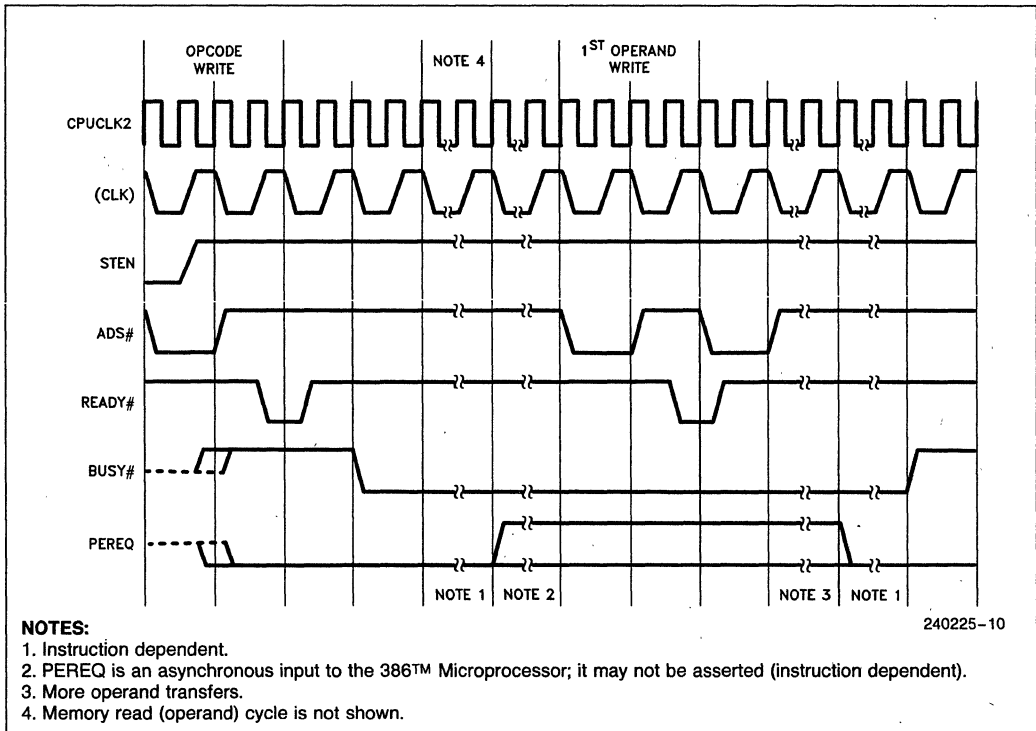


Figure 4-5. STEN, BUSY #, and PEREQ Timing Relationships

5.0 PACKAGE THERMAL SPECIFICATIONS

The 387 SX Math Coprocessor is specified for operation when case temperature is within the range of 0°C–100°C. The case temperature may be measured in any environment, to determine whether the 387 SX Math Coprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature is guaranteed as long as T_c is not violated. The ambient temperature can be calculated from the θ_{jc} and θ_{ja} from the following equations:

$$T_j = T_c + P * \theta_{jc}$$

$$T_a = T_j - P * \theta_{ja}$$

$$T_c = T_a + P * [\theta_{ja} - \theta_{jc}]$$

Values for θ_{ja} and θ_{jc} are given in Table 5-1 for the 68-pin PLCC. θ_{ja} is given at various airflows. Table 5-2 shows the maximum T_a allowable (without exceeding T_c) at various airflows. Note that T_a can be improved further by attaching 'fins' or a 'heat sink' to the package. P is calculated by using the maximum hot I_{CC} .

Table 5-1. Thermal Resistances (°C/Watt) θ_{jc} and θ_{ja}

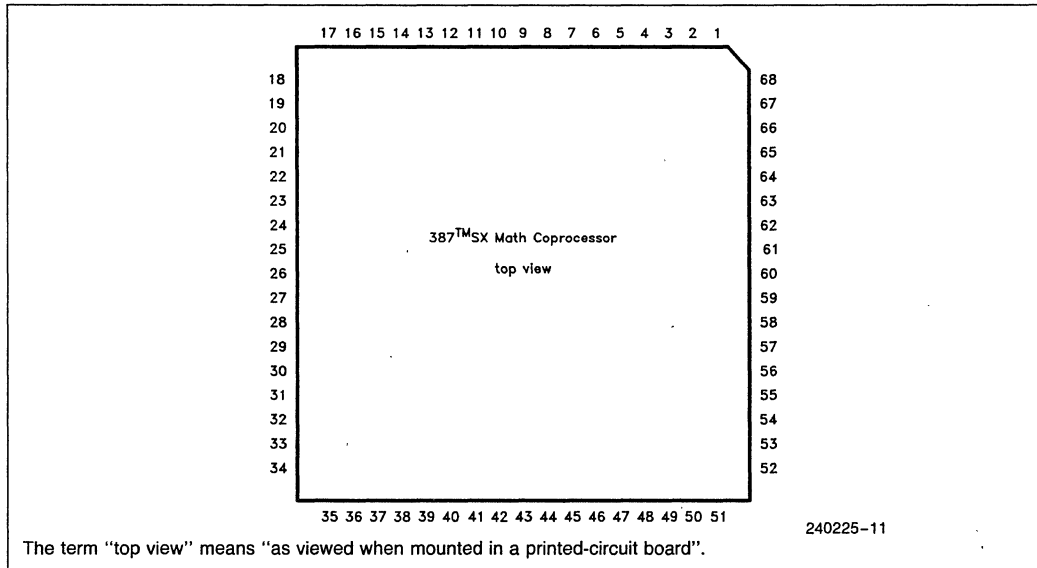
Package	θ_{jc}	θ_{ja} versus Airflow - ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
68-Pin PLCC	8	30	25	20	15.5	13	12

Table 5-2. Maximum T_A at Various Airflows

Package	T_A (°C) versus Airflow - ft/min (m/sec)					
	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
68-Pin PLCC	54.7	61.6	68.5	74.6	78.1	79.5

Max. T_A calculated at Max V_{CC} and Max I_{CC} .

Figure 5-1 shows the locations of pins on the chip package. Table 5-3 helps to locate pin identifiers in Figure 5-1.


Figure 5-1. PLCC Pin Configuration
Table 5-3. Pin Cross-Reference

1 — n.c.	18 — n.c.	35 — ERROR #	52 — n.c.
2 — D07	19 — D00	36 — BUSY #	53 — NUMCLK2
3 — D06	20 — D01	37 — V_{CC}	54 — CPUCLK2
4 — V_{CC}	21 — V_{SS}	38 — V_{SS}	55 — V_{SS}
5 — V_{SS}	22 — V_{CC}	39 — V_{CC}	56 — PEREQ
6 — D05	23 — D02	40 — STEN	57 — READY #
7 — D04	24 — D08	41 — W/R #	58 — V_{CC}
8 — D03	25 — V_{SS}	42 — V_{SS}	59 — CKM
9 — V_{CC}	26 — V_{CC}	43 — V_{CC}	60 — V_{SS}
10 — n.c.	27 — V_{SS}	44 — NPS1 #	61 — V_{SS}
11 — D15	28 — D09	45 — NPS2	62 — V_{CC}
12 — D14	29 — D10	46 — V_{CC}	63 — V_{SS}
13 — V_{CC}	30 — D11	47 — ADS #	64 — V_{CC}
14 — V_{SS}	31 — V_{CC}	48 — CMD0 #	65 — n.c.
15 — D13	32 — V_{SS}	49 — READY #	66 — V_{SS}
16 — D12	33 — V_{CC}	50 — V_{CC}	67 — n.c.
17 — n.c.	34 — V_{SS}	51 — RESETIN	68 — n.c.

n.c.—The corresponding pins of the 387™ SX NPX are left unconnected.

6.0 ELECTRICAL DATA

6.1 Absolute Maximum Ratings

NOTE:

Stresses above those listed may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the

operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Case temperature T_C under bias 0°C to 100°C
 Storage temperature -65°C to $+150^{\circ}\text{C}$
 Voltage on any pin with respect to ground -0.5 to $V_{CC}+0.5\text{V}$
 Power dissipation 1.5 Watt

6.2 D.C. Characteristics

Table 6-1. D.C. Specifications $T_C = 0^{\circ}$ to 100°C , $V_{CC} = 5\text{V} \pm 10\%$

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input LO Voltage	-0.3	+0.8	V	See note 1
V_{IH}	Input HI Voltage	2.0	$V_{CC}+0.3$	V	See note 1
V_{CL}	CPUCLK2 and NUMCLK2 Input LO Voltage	-0.3	+0.8	V	
V_{CH}	CPUCLK2 and NUMCLK2 Input HI Voltage	$V_{CC}-0.8$	$V_{CC}+0.3$	V	
V_{OL}	Output LO Voltage		0.45	V	See note 2
V_{OH}	Output HI Voltage	2.4		V	See note 3
V_{OH}	Output HI Voltage	$V_{CC}-0.8$		V	See note 4
I_{CC}	Power Supply Current NUMCLK2 = 32 MHz ⁽⁵⁾ NUMCLK2 = 2 MHz ⁽⁵⁾		250 100	mA mA	I_{CC} typ. = 150 mA
I_{LI}	Input Leakage Current		± 15	μA	$0\text{V} \leq V_{IN} \leq V_{CC}$
I_{LO}	I/O Leakage Current		± 15	μA	$0.45\text{V} \leq V_O \leq V_{CC}$
C_{IN}	Input Capacitance		10	pF	$f_c = 1\text{MHz}$
C_O	I/O or Output Capacitance		12	pF	$f_c = 1\text{MHz}$
C_{CLK}	Clock Capacitance		20	pF	$f_c = 1\text{MHz}$

NOTES:

- This parameter is for all inputs, excluding the clock inputs.
- This parameter is measured at I_{OL} as follows:
data = 4.0mA
READY0#, ERROR#, BUSY#, PEREQ = 2.5mA
- This parameter is measured at I_{OH} as follows:
data = 1.0mA
READY0#, ERROR#, BUSY#, PEREQ = 0.6mA
- This parameter is measured at I_{OH} as follows:
data = 0.2mA
READY0#, ERROR#, BUSY#, PEREQ = 0.12mA
- I_{CC} is measured at steady state, maximum capacitive loading on the outputs, and worst-case D.C. level at the inputs; CPUCLK2 at the same frequency as NUMCLK2.

6.3 A.C. Characteristics

Table 6-2a. Combinations of Bus Interface and Execution Speeds

Functional Block	80387SX-16
Bus Interface Unit (MHz)	16
Execution Unit (MHz)	16

Table 6-2b. Timing Requirements of Execution Unit $T_C = 0^\circ$ to 100° C, $V_{CC} = 5V \pm 10\%$

Pin	Symbol	Parameter	16 MHz		Test Conditions	Refer to Figure
			Min (ns)	Max (ns)		
NUMCLK2	t1	Period	31.25	500	2.0V	6.2 (Note 1)
NUMCLK2	t2b	High Time	5		$V_{CC} - 0.8V$	
NUMCLK2	t3b	Low Time	7		0.8V	
NUMCLK2	t4	Fall Time		8	From $V_{CC} - 0.8$ to 0.8V	
NUMCLK2	t5	Rise Time		8	From 0.8 to $V_{CC} - 0.8V$	

Note:

1. If not used (CKM = 1), tie LOW.

Table 6-2c. Timing Requirements of Bus Interface Unit $T_C = 0^\circ$ to 100° C, $V_{CC} = 5V \pm 10\%$

Pin	Symbol	Parameter	16 MHz (1.5V)		Test Conditions	Refer to Figure
			Min (ns)	Max (ns)		
CPUCLK2	t1	Period	31.25	500	2.0V	6.2
CPUCLK2	t2b	High Time	5		$V_{CC} - 0.8V$	
CPUCLK2	t3b	Low Time	7		0.8V	
CPUCLK2	t4	Fall Time		8	From $V_{CC} - 0.8$ to 0.8V	
CPUCLK2	t5	Rise Time		8	From 0.8 to $V_{CC} - 0.8V$	
CPUCLK2/ NUMCLK2		Ratio	10/16	16/10		
READYO#	t7	Out Delay	4	34	$C_L = 75pf$	6.3
READYO#	t7	Out Delay	4	31	$C_L = 25pf^{**}$	
PEREQ	t7	Out Delay	5	34	$C_L = 75pf$	
BUSY#	t7	Out Delay	5	34	$C_L = 75pf$	
ERROR#	t7	Out Delay	5	34	$C_L = 75pf$	
D15-D0	t8	Out Delay		54	$C_L = 120pf$	6.4
D15-D0	t10	Setup Time	11			
D15-D0	t11	Hold Time	11			
D15-D0	t12*	Float Time	6	33	$C_L = 120pf$	
PEREQ	t13*	Float Time	1	60	$C_L = 75pf$	6.6
BUSY#	t13*	Float Time	1	60	$C_L = 75pf$	
ERROR#	t13*	Float Time	1	60	$C_L = 75pf$	
READYO#	t13*	Float Time	1	60	$C_L = 75pf$	
ADS#	t14	Setup Time	26			6.4
ADS#	t15	Hold Time	5			
W/R#	t14	Setup Time	26			
W/R#	t15	Hold Time	5			
READY#	t16	Setup Time	19			6.4
READY#	t17	Hold Time	4			
CMD0#	t16	Setup Time	21			
CMD0#	t17	Hold Time	2			
NPS1#, NPS2	t16	Setup Time	21			
NPS1#, NPS2	t17	Hold Time	2			
STEN	t16	Setup Time	21			
STEN	t17	Hold Time	2			
RESETIN	t18	Setup Time	13			
RESETIN	t19	Hold Time	4			

NOTES:

 *Float condition occurs when maximum output current becomes less than the I_{LO} in magnitude. Float delay is not tested.

**Not tested at 25 pf.

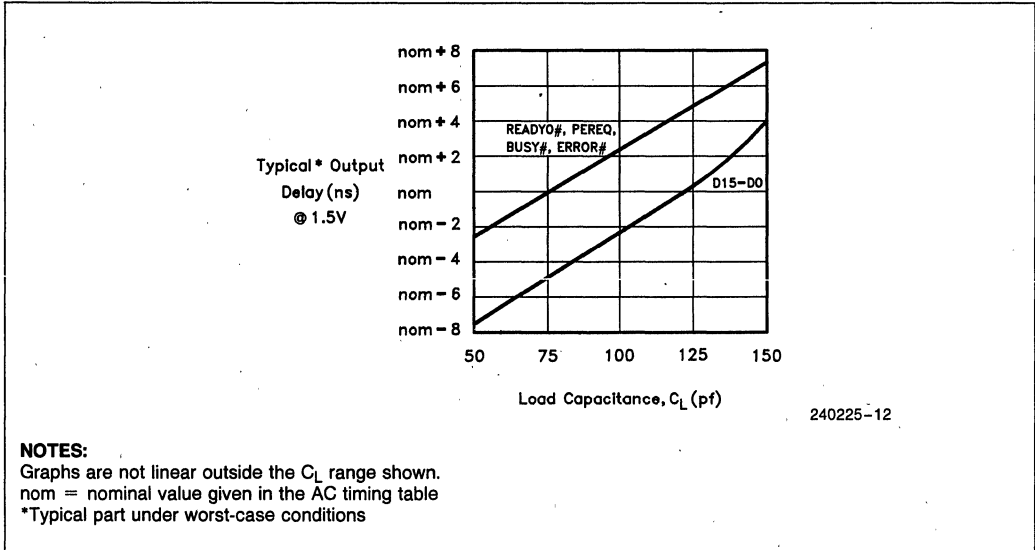


Figure 6-1a. Typical Output Valid Delay vs. Load Capacitance at Max Operating Temperature

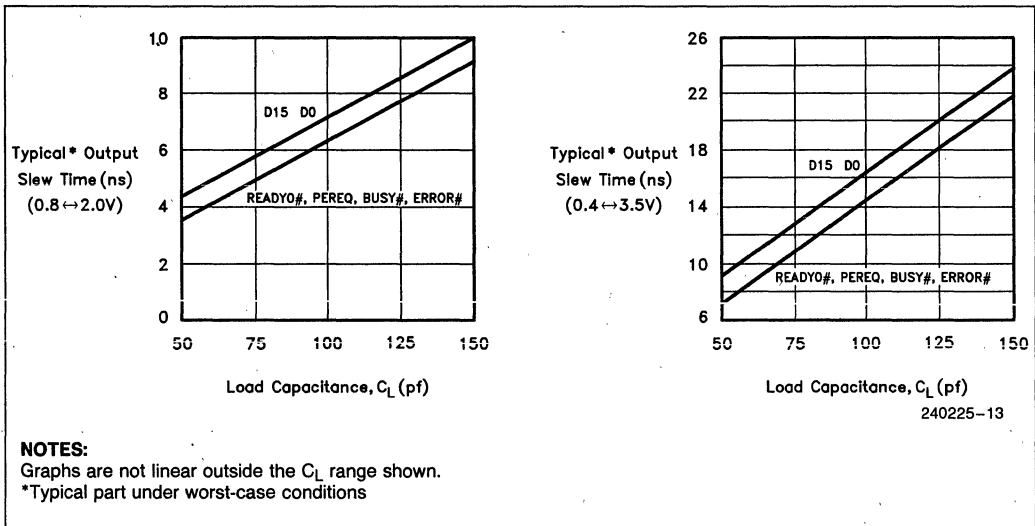
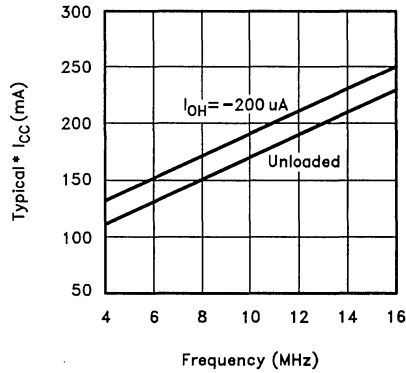


Figure 6-1b. Typical Output Slew Time vs. Load Capacitance at Max Operating Temperature

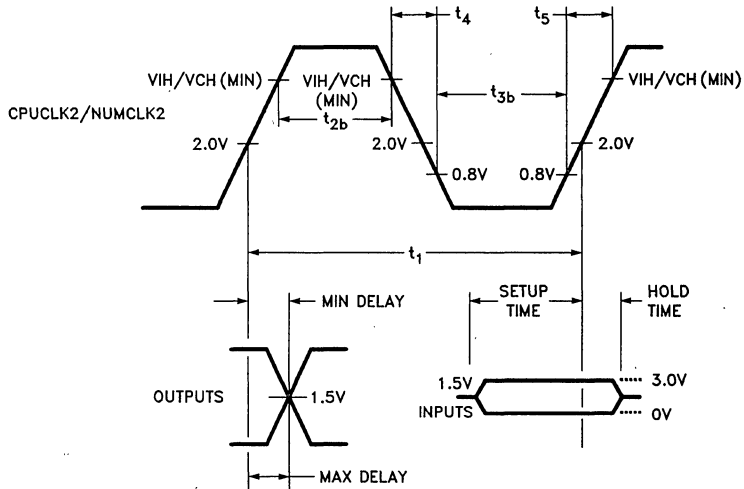


240225-14

NOTES:

Graphs are not linear outside the frequency range shown.
 *Typical part under worst-case conditions.

Figure 6-1c. Typical I_{CC} vs. Load Capacitance at Max Operating Temperature



240225-15

Figure 6-2. CPUCLK2/NUMCLK2 Waveform and Measurement Points for Input/Output

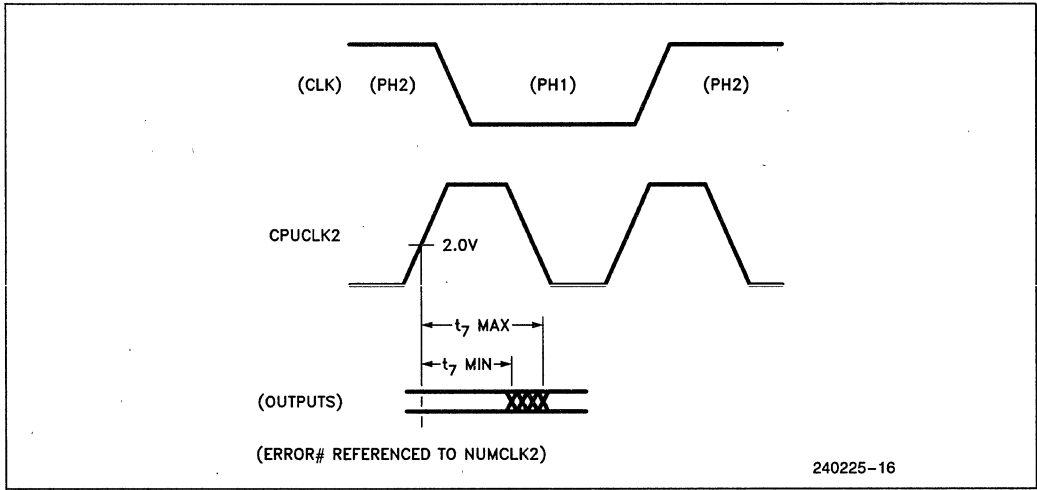


Figure 6-3. Output Signals

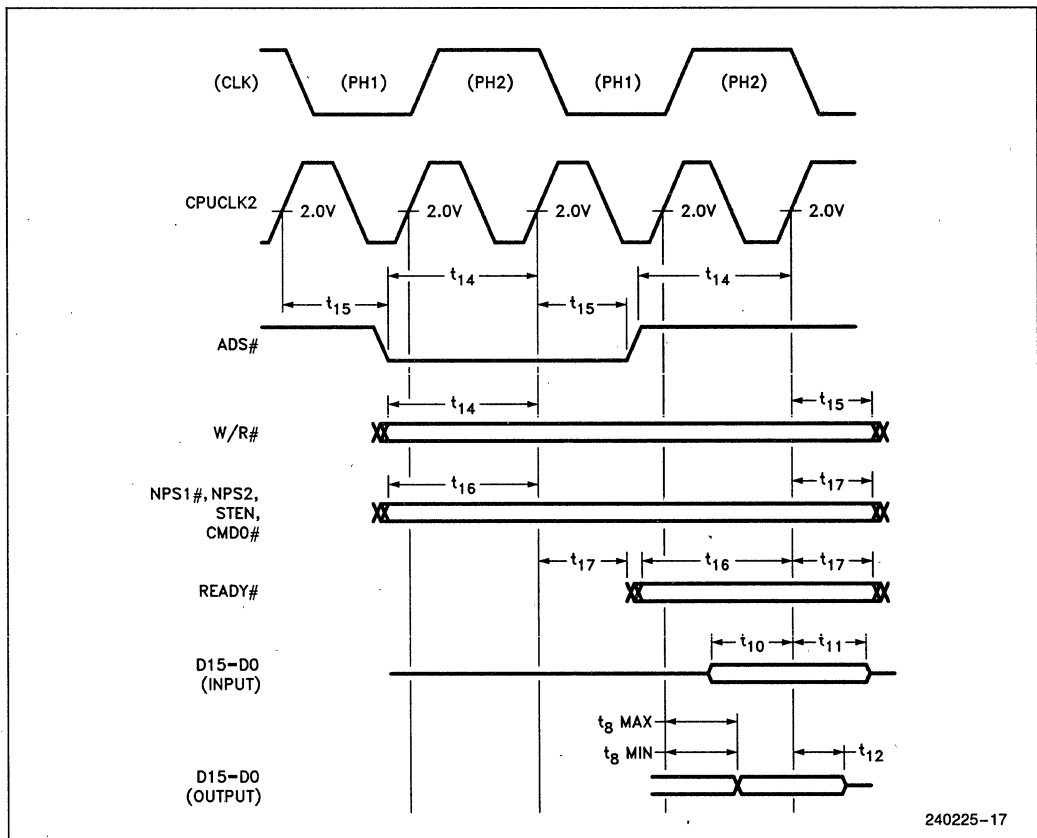


Figure 6-4. Input and I/O Signals

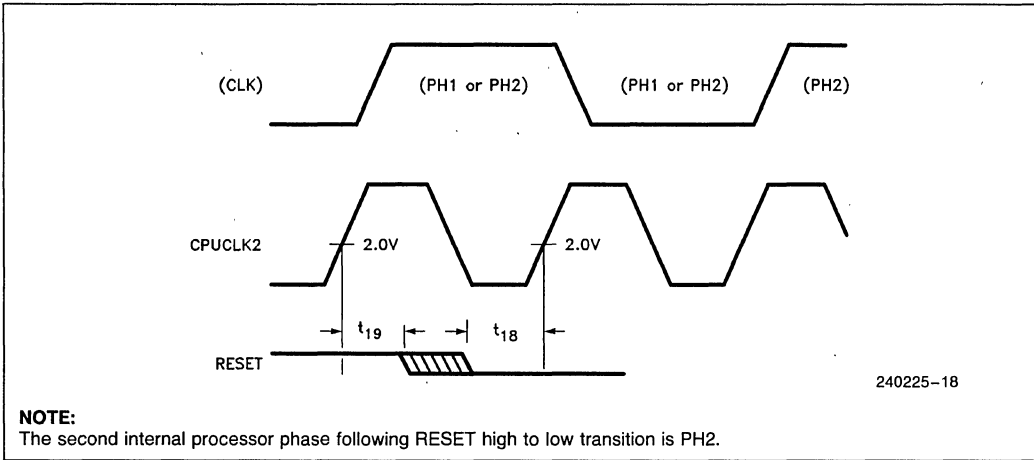


Figure 6-5. RESET Signal

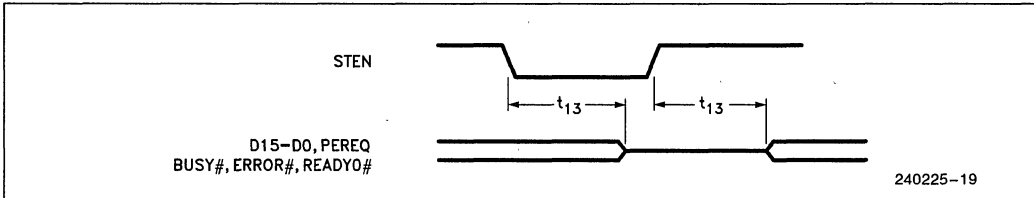


Figure 6-6. Float from STEN

Table 6-3. Other Parameters

Pin	Symbol	Parameter	Min	Max	Units
RESETIN	t30	Duration	40		NUMCLK2
RESETIN	t31	RESETIN inactive to 1st opcode write	50		NUMCLK2
BUSY #	t32	Duration	6		CPUCLK2
BUSY #, ERROR #	t33	ERROR # (in)active to BUSY # inactive	6		CPUCLK2
PEREQ, ERROR #	t34	PEREQ inactive to ERROR # active	6		CPUCLK2
READY #, BUSY #	t35	READY # active to BUSY # active	4	4	CPUCLK2
READY #	t36	Minimum time from opcode write to opcode/operand write	4		CPUCLK2
READY #	t37	Minimum time from operand write to operand write	4		CPUCLK2

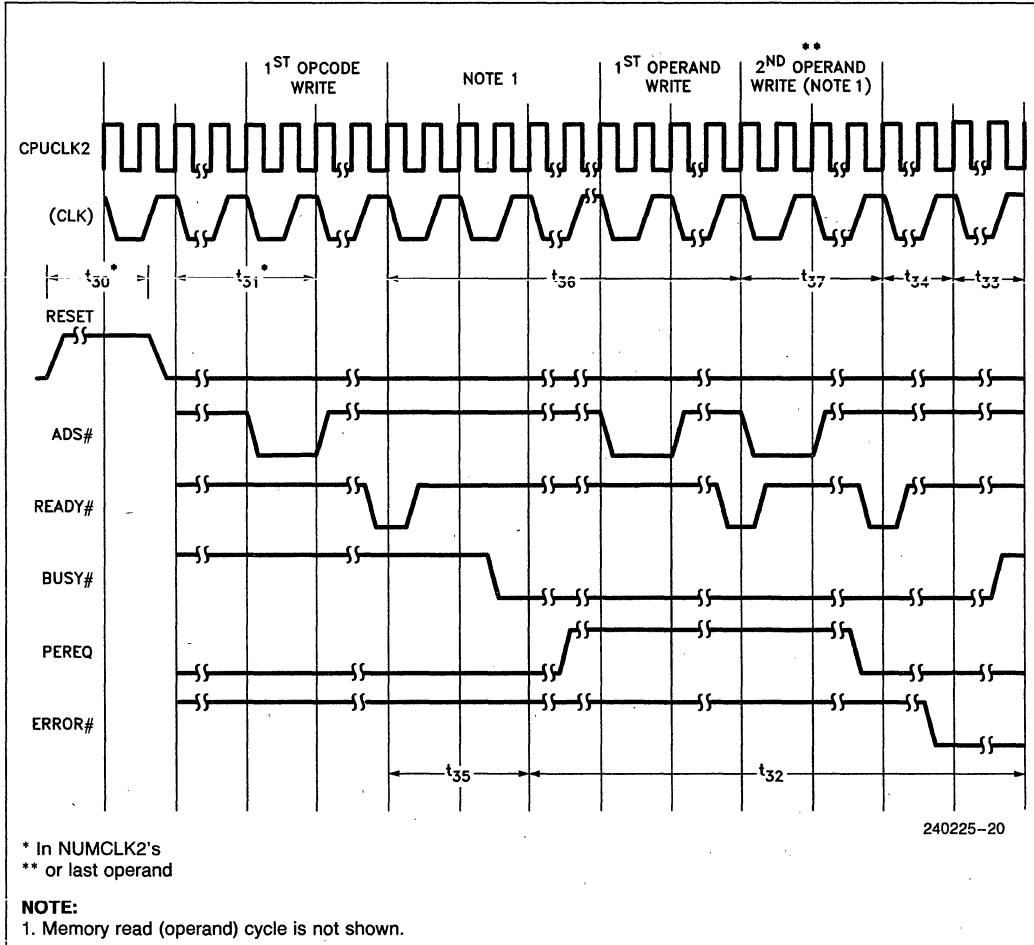


Figure 6-7. Other Parameters

7.0 387™ SX NPX EXTENSIONS TO THE CPU'S INSTRUCTION SET

Instructions for the 387 SX NPX assume one of the five forms shown in Table 7-1. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B, which identifies the ESCAPE class of instruction. Instructions that refer to memory operands specify addresses using the CPU's addressing modes.

MOD (Mode field) and R/M (Register/Memory specifier) have the same interpretation as the corresponding fields of CPU instructions (refer to Programmer's Reference Manual for the CPU). SIB

(Scale Index Base) byte and DISP (displacement) are optionally present in instructions that have MOD and R/M fields. Their presence depends on the values of MOD and R/M, as for instructions of the CPU.

The instruction summaries that follow assume that the instruction has been prefetched, decoded, and is ready for execution; that bus cycles do not require wait states; that there are no local bus HOLD requests delaying processor access to the bus; and that no exceptions are detected during instruction execution. If the instruction has MOD and R/M fields that call for both base and index registers, add one clock.

Table 7-1. Instruction Formats

		Instruction							Optional Fields		
		First Byte			Second Byte						
1	11011	OPA		1	MOD	1	OPB	R/M	SIB	DISP	
2	11011	MF			OPA	MOD	OPB*		R/M	SIB	DISP
3	11011	d	P	OPA	1	1	OPB*		ST(i)		
4	11011	0	0	1	1	1	1	OP			
5	11011	0	1	1	1	1	1	OP			

15-11 10 9 8 7 6 5 4 3 2 1 0

OP = Instruction opcode, possibly split into two fields OPA and OPB

MF = Memory Format

00—32-bit real

01—32-bit integer

10—64-bit real

11—16-bit integer

d = Destination

0—Destination is ST(0)

1—Destination is ST(i)

R XOR d = 0—Destination (op) Source

R XOR d = 1—Source (op) Destination

*In FSUB and FDIV, the low-order bit of OPB is the R (reversed) bit

P = POP

0—Do not pop stack

1—Pop stack after operation

ESC = 11011

ST(i) = Register stack element i

000 = Stack top

001 = Second stack element

.

.

.

111 = Eighth stack element

387™ SX NPX Extension to the 386™ SX Microprocessor Instruction Set

Instruction	Encoding			Clock Count Range			
	Byte 0	Byte 1	Optional Bytes 2-6	32-Bit Real	32-Bit Integer	64-Bit Real	16-Bit Integer
DATA TRANSFER							
FLD = Load ^a							
Integer/real memory to ST(0)	ESC MF 1	MOD 000 R/M	SIB/DISP	24	49-56	33	61-65
Long integer memory to ST(0)	ESC 111	MOD 101 R/M	SIB/DISP		64-75		
Extended real memory to ST(0)	ESC 011	MOD 101 R/M	SIB/DISP		52		
BCD memory to ST(0)	ESC 111	MOD 100 R/M	SIB/DISP		274-283		
ST(i) to ST(0)	ESC 001	11000 ST(i)			14		
FST = Store							
ST(0) to integer/real memory	ESC MF 1	MOD 010 R/M	SIB/DISP	49	84-98	55	82-95
ST(0) to ST(i)	ESC 101	11010 ST(i)			11		
FSTP = Store and Pop							
ST(0) to integer/real memory	ESC MF 1	MOD 011 R/M	SIB/DISP	49	84-98	55	82-95
ST(0) to long integer memory	ESC 111	MOD 111 R/M	SIB/DISP		90-107		
ST(0) to extended real	ESC 011	MOD 111 R/M	SIB/DISP		63		
ST(0) to BCD memory	ESC 111	MOD 110 R/M	SIB/DISP		522-544		
ST(0) to ST(i)	ESC 101	11011 ST(i)			12		
FXCH = Exchange							
ST(i) and ST(0)	ESC 001	11001 ST(i)			18		
COMPARISON							
FCOM = Compare							
Integer/real memory to ST(0)	ESC MF 0	MOD 010 R/M	SIB/DISP	30	60-67	39	71-75
ST(i) to ST(0)	ESC 000	11010 ST(i)			24		
FCOMP = Compare and pop							
Integer/real memory to ST	ESC MF 0	MOD 011 R/M	SIB/DISP	30	60-67	39	71-75
ST(i) to ST(0)	ESC 000	11011 ST(i)			26		
FCOMPP = Compare and pop twice							
ST(1) to ST(0)	ESC 110	1101 1001			26		
FTST = Test ST(0)							
	ESC 001	1110 0100			28		
FUCOM = Unordered compare							
	ESC 101	11100 ST(i)			24		
FUCOMP = Unordered compare and pop							
	ESC 101	11101 ST(i)			26		
FUCOMPP = Unordered compare and pop twice							
	ESC 010	1110 1001			26		
FXAM = Examine ST(0)							
	ESC 001	11100101			30-38		
CONSTANTS							
FLDZ = Load +0.0 into ST(0)							
	ESC 001	1110 1110			20		
FLD1 = Load +1.0 into ST(0)							
	ESC 001	1110 1000			24		
FLDPI = Load pi into ST(0)							
	ESC 001	1110 1011			40		
FLDL2T = Load log ₂ (10) into ST(0)							
	ESC 001	1110 1001			40		

Shaded areas indicate instructions not available in 8087/80287.

NOTE:

a. When loading single- or double-precision zero from memory, add 5 clocks.

387™ SX NPX Extension to the 386™ SX Microprocessor Instruction Set (Continued)

Instruction	Encoding			Clock Count Range			
	Byte 0	Byte 1	Optional Bytes 2-6	32-Bit Real	32-Bit Integer	64-Bit Real	16-Bit Integer
CONSTANTS (Continued)							
FLDL2E = Load $\log_2(e)$ into ST(0)	ESC 001	1110 1010			40		
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	ESC 001	1110 1100			41		
FLDLN2 = Load $\log_e(2)$ into ST(0)	ESC 001	1110 1101			41		
ARITHMETIC							
FADD = Add							
Integer/real memory with ST(0)	ESC MF 0	MOD 000 R/M	SIB/DISP	28-36	61-76	37-45	71-85
ST(i) and ST(0)	ESC d P 0	11000 ST(i)			23-31 ^b		
FSUB = Subtract							
Integer/real memory with ST(0)	ESC MF 0	MOD 10 R R/M	SIB/DISP	28-36	61-76	36-44	71-83 ^c
ST(i) and ST(0)	ESC d P 0	1110 R R/M			26-34 ^d		
FMUL = Multiply							
Integer/real memory with ST(0)	ESC MF 0	MOD 001 R R/M	SIB/DISP	31-39	65-86	40-65	76-87
ST(i) and ST(0)	ESC d P 0	1100 1 R/M			29-57 ^e		
FDIV = Divide							
Integer/real memory with ST(0)	ESC MF 0	MOD 11 R R/M	SIB/DISP	93	124-131 ^f	102	136-140 ^g
ST(i) and ST(0)	ESC d P 0	1111 R R/M			88 ^h		
FSQRT ⁱ = Square root	ESC 001	1111 1010			122-129		
FSCALE = Scale ST(0) by ST(1)	ESC 001	1111 1101			67-86		
FPREM = Partial remainder	ESC 001	1111 1000			74-155		
FPREM1 = Partial remainder (IEEE)	ESC 001	1111 0101			95-185		
FRNDINT = Round ST(0) to integer	ESC 001	1111 1100			66-80		
FXTRACT = Extract components of ST(0)	ESC 001	1111 0100			70-76		
FABS = Absolute value of ST(0)	ESC 001	1110 0001			22		
FNCHS = Change sign of ST(0)	ESC 001	1110 0000			24-25		

Shaded areas indicate instructions not available in 8087/80287.

NOTES:

- b. Add 3 clocks to the range when $d = 1$.
- c. Add 1 clock to **each** range when $R = 1$.
- d. Add 3 clocks to the range when $d = 0$.
- e. typical = 52 (When $d = 0$, 46-54, typical = 49).
- f. Add 1 clock to the range when $R = 1$.
- g. 135-141 when $R = 1$.
- h. Add 3 clocks to the range when $d = 1$.
- i. $-0 \leq ST(0) \leq +\infty$.

387™ SX NPX Extension to the 386™ SX Microprocessor Instruction Set (Continued)

Instruction	Encoding			Clock Count Range
	Byte 0	Byte 1	Optional Bytes 2-6	
TRANSCENDENTAL				
FCOS^k = Cosine of ST(0)	ESC 001	1111 1111		123-772
FP_{TAN}^k = Partial tangent of ST(0)	ESC 001	1111 0010		191-497
FP_{ATAN} = Partial arctangent	ESC 001	1111 0011		314-487
FSIN^k = Sine of ST(0)	ESC 001	1111 1110		122-771
FSINCOS^k = Sine and cosine of ST(0)	ESC 001	1111 1011		194-809
F2XM1^l = 2 ^{ST(0)} - 1	ESC 001	1111 0000		211-476
FYL2X^m = ST(1) * log ₂ (ST(0))	ESC 001	1111 0001		120-538
FYL2XP1ⁿ = ST(1) * log ₂ (ST(0) + 1.0)	ESC 001	1111 1001		257-547
PROCESSOR CONTROL				
FINIT = Initialize NPX	ESC 011	1110 0011		33
FSTSW AX = Store status word	ESC 111	1110 0000		13
FLDCW = Load control word	ESC 001	MOD 101 R/M	SIB/DISP	19
FSTCW = Store control word	ESC 101	MOD 111 R/M	SIB/DISP	15
FSTSW = Store status word	ESC 101	MOD 111 R/M	SIB/DISP	15
FCLEX = Clear exceptions	ESC 011	1110 0010		11
FSTENV = Store environment	ESC 001	MOD 110 R/M	SIB/DISP	103-104
FLDENV = Load environment	ESC 001	MOD 100 R/M	SIB/DISP	71
FSAVE = Save state	ESC 101	MOD 110 R/M	SIB/DISP	475-476
FRSTOR = Restore state	ESC 101	MOD 100 R/M	SIB/DISP	388
FINCSTP = Increment stack pointer	ESC 001	1111 0111		21
FDECSTP = Decrement stack pointer	ESC 001	1111 0110		22
FFREE = Free ST(i)	ESC 101	1100 0 ST(i)		18
FNOP = No operations	ESC 001	1101 0000		12

Shaded areas indicate instructions not available in 8087/80287.

NOTES:

j. These timings hold for operands in the range $|x| < \pi/4$. For operands not in this range, up to 76 additional clocks may be needed to reduce the operand.

k. $0 \leq |ST(0)| < 2^{63}$.

l. $-1.0 \leq ST(0) \leq 1.0$.

m. $0 \leq ST(0) < \infty$, $-\infty < ST(1) < +\infty$.

n. $0 \leq |ST(0)| < (2 - \text{SQRT}(2))/2$, $-\infty < ST(1) < +\infty$.

APPENDIX A COMPATIBILITY BETWEEN THE 80287 AND THE 8087

The 80286/80287 operating in Real-Address mode will execute 8086/8087 programs without major modification. However, because of differences in the handling of numeric exceptions by the 80287 NPX and the 8087 NPX, exception-handling routines *may* need to be changed.

This appendix summarizes the differences between the 80287 NPX and the 8087 NPX, and provides details showing how 8086/8087 programs can be ported to the 80286/80287.

1. The NPX signals exceptions through a dedicated ERROR# line to the 80286. The NPX error signal does not pass through an interrupt controller (the 8087 INT signal does). Therefore, any interrupt-controller-oriented instructions in numeric exception handlers for the 8086/8087 should be deleted.
2. The 8087 instructions FENI/FNENI and FDISI/FNDISI perform no useful function in the 80287. If the 80287 encounters one of these opcodes in its instruction stream, the instruction will effectively be ignored—none of the 80287 internal states will be updated. While 8086/8087 containing these instructions may be executed on the 80286/80287, it is unlikely that the exception-handling routines containing these instructions will be completely portable to the 80287.
3. Interrupt vector 16 must point to the numeric exception handling routine.
4. The ESC instruction address saved in the 80287 includes any leading prefixes before the ESC opcode. The corresponding address saved in the 8087 does not include leading prefixes.
5. In Protected-Address mode, the format of the 80287's saved instruction and address pointers is different than for the 8087. The instruction opcode is not saved in Protected mode—exception handlers will have to retrieve the opcode from memory if needed.
6. Interrupt 7 will occur in the 80286 when executing ESC instructions with either TS (task switched) or EM (emulation) of the 80286 MSW set (TS = 1 or EM = 1). If TS is set, then a WAIT instruction will also cause interrupt 7. An exception handler should be included in 80286/80287 code to handle these situations.
7. Interrupt 9 will occur if the second or subsequent words of a floating-point operand fall outside a segment's size. Interrupt 13 will occur if the starting address of a numeric operand falls outside a segment's size. An exception handler should be included in 80286/80287 code to report these programming errors.
8. Except for the processor control instructions, all of the 80287 numeric instructions are automatically synchronized by the 80286 CPU—the 80286 automatically tests the BUSY# line from the 80287 to ensure that the 80287 has completed its previous instruction before executing the next ESC instruction. No explicit WAIT instructions are required to assure this synchronization. For the 8087 used with 8086 and 8088 processors, explicit WAITs are required before each numeric instruction to ensure synchronization. Although 8086/8087 programs having explicit WAIT instructions will execute perfectly on the 80286/80287 without reassembly, these WAIT instructions are unnecessary.
9. Since the 80287 does not require WAIT instructions before each numeric instruction, the ASM286 assembler does not automatically generate these WAIT instructions. The ASM86 assembler, however, automatically precedes every ESC instruction with a WAIT instruction. Although numeric routines generated using the ASM86 assembler will generally execute correctly on the 80286/80287, reassembly using ASM286 may result in a more compact code image.

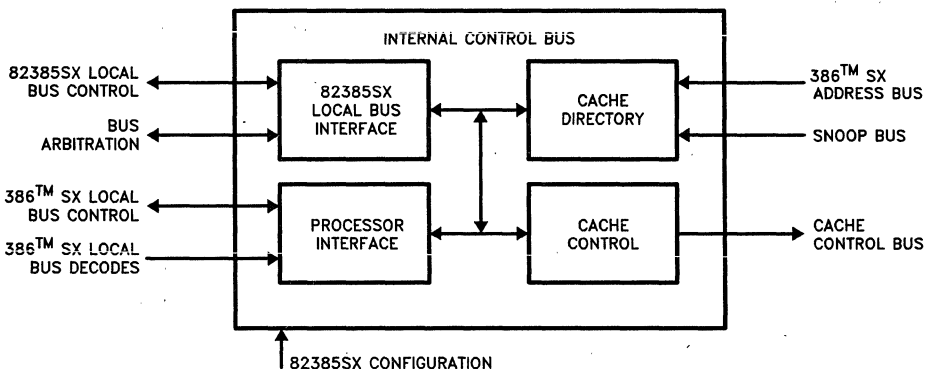
The processor control instructions for the 80287 may be coded using either a WAIT or No-WAIT form of mnemonic. The WAIT forms of these instructions cause ASM286 to precede the ESC instruction with a CPU WAIT instruction, in the identical manner as does ASM86.

82385SX HIGH PERFORMANCE CACHE CONTROLLER

- **Improves 386™ SX System Performance**
 - Reduces Average CPU Wait States to Nearly Zero
 - Zero Wait State Read Hit
 - Zero Wait State Posted Memory Writes
 - Allows Other Masters to Access the System Bus More Readily
- **Hit Rates up to 99%**
- **Optimized as 386 SX Companion**
 - Simple 386 SX Interface
 - Part of Intel386™-Based Compute Engine Including 387™ SX Math Coprocessor and 82370 Integrated System Peripheral
 - 16 MHz Operation
- **Software Transparent**
- **Synchronous Dual Bus Architecture**
 - Bus Watching Maintains Cache Coherency
- **Maps Full 386 SX Address Space**
- **Flexible Cache Mapping Policies**
 - Direct Mapped or 2-Way Set Associative Cache Organization
 - Supports Non-Cacheable Memory Space
 - Unified Cache for Code and Data
- **Integrates Cache Directory and Cache Management Logic**
- **High Speed CHMOS Technology**
 - 132-Pin PGA and 132-Lead PQFP

The 82385SX Cache Controller is a high performance peripheral for Intel's 386™ SX Microprocessor. It stores a copy of frequently accessed code and data from main memory in a zero wait state local cache memory. The 82385SX allows the 386 SX Microprocessor to run near its full potential by reducing the average number of CPU wait states to nearly zero. The dual bus architecture of the 82385SX allows other masters to access system resources while the 386 SX CPU operates locally out of its cache. In this situation, the 82385SX's "bus watching" mechanism preserves cache coherency by monitoring the system bus address lines at no cost to system or local throughput.

The 82385SX is completely software transparent, protecting the integrity of system software. High performance and board space savings are achieved because the 82385SX integrates a cache directory and all cache management logic on one chip.



82385SX Internal Block Diagram

290222-1

1.0 82385SX FUNCTIONAL OVERVIEW

The 82385SX Cache Controller is a high performance peripheral for Intel's 386™ SX microprocessor. This chapter provides an overview of the 82385SX, and of the basic architecture and operation of a 386 SX CPU/82385SX system.

1.1 82385SX Overview

The main function of a cache memory system is to provide fast local storage for frequently accessed code and data. The cache system intercepts 386 SX memory references to see if the required data resides in the cache. If the data resides in the cache (a hit), it is returned to the 386 SX without incurring wait states. If the data is not cached (a miss), the reference is forwarded to the system and the data retrieved from main memory. An efficient cache will yield a high "hit rate" (the ratio of cache hits to total 386 SX accesses), such that the majority of accesses are serviced with zero wait states. The net effect is that the wait states incurred in a relatively infrequent miss are averaged over a large number of accesses, resulting in an average of nearly zero wait states per access. Since cache hits are serviced locally, a processor operating out of its local cache has a much lower "bus utilization" which reduces system bus bandwidth requirements, making more bandwidth available to other bus masters.

The 82385SX Cache Controller integrates a cache directory and all cache management logic required to support an external 16 kbyte cache. The cache directory structure is such that the entire physical address range of the 386 SX is mapped into the cache. Provision is made to allow areas of memory to be set aside as non-cacheable. The user has two cache organization options: direct mapped and 2-way set associative. Both provide the high hit rates necessary to make a large, relatively slow main memory array look like fast, zero wait state memory to the 386 SX.

A good hit rate is an essential ingredient of a successful cache implementation. Hit rate is the measure of how efficient a cache is in maintaining a copy of the most frequently requested code and data. However, efficiency is not the only factor for performance consideration. Just as essential are sound cache management policies. These policies refer to the handling of 386 SX writes, preservation of cache coherency, and ease of system design. The 82385SX's "posted write" capability allows the majority of 386 SX writes, including most non-cacheable cycles, to run with zero wait states, and the 82385SX's "bus watching" mechanism preserves

cache coherency with no impact on system performance. Physically, the 82385SX ties directly to the 386 SX with virtually no external logic.

1.2 System Overview I: Bus Structure

A good grasp of bus structure of a 386 SX CPU/82385SX system is essential in understanding both the 82385SX and its role in a 386 SX system. The following is a description of this structure.

1.2.1 386™ SX LOCAL BUS/82385SX LOCAL BUS/SYSTEM BUS

Figure 1-1 depicts the bus structure of a typical 386 SX system. The "386 SX Local Bus" consists of the physical 386 SX address, data, and control buses. The local address and data busses are buffered and/or latched to become the "system" address and data busses. The local control bus is decoded by bus control logic to generate the various system bus read and write commands.

The addition of an 82385SX Cache Controller causes a separation of the 386 SX bus into two distinct busses: the actual 386 SX local bus and the "82385SX Local Bus" (Figure 1-2). The 82385SX local bus is designed to look like the front end of a 386 SX by providing 82385SX local bus equivalents to all appropriate 386 SX signals. The system ties to this "386 SX-like" front end just as it would to an actual 386 SX. The 386 SX simply sees a fast system bus, and the system sees a 386 SX front end with low bus bandwidth requirements. The cache subsystem is transparent to both. Note that the 82385SX local bus is not simply a buffered version of the 386 SX bus, but rather is distinct from, and able to operate in parallel with the 386 SX bus. Other masters residing on either the 82385SX local bus or system bus are free to manage system resources while the 386 SX operates out of its cache.

1.2.2 BUS ARBITRATION

The 82385SX presents the "386 SX-like" interface which is called the 82385SX local bus. Whereas the 386 SX provides a Hold Request/ Hold Acknowledge bus arbitration mechanism via its HOLD and HLDA pins, the 82385SX provides an equivalent mechanism via its BHOLD and BHLDA pins. (These signals are described in Section 3.7.) When another master requests the 82385SX local bus, it issues the request to the 82385SX via BHOLD. Typically, at the end of the current 82385SX local bus cycle, the 82385SX will release the 82385SX local bus and acknowledge the request via BHLDA. The 386 SX is of course free to continue operating on the 386 SX local bus while another master owns the 82385SX local bus.

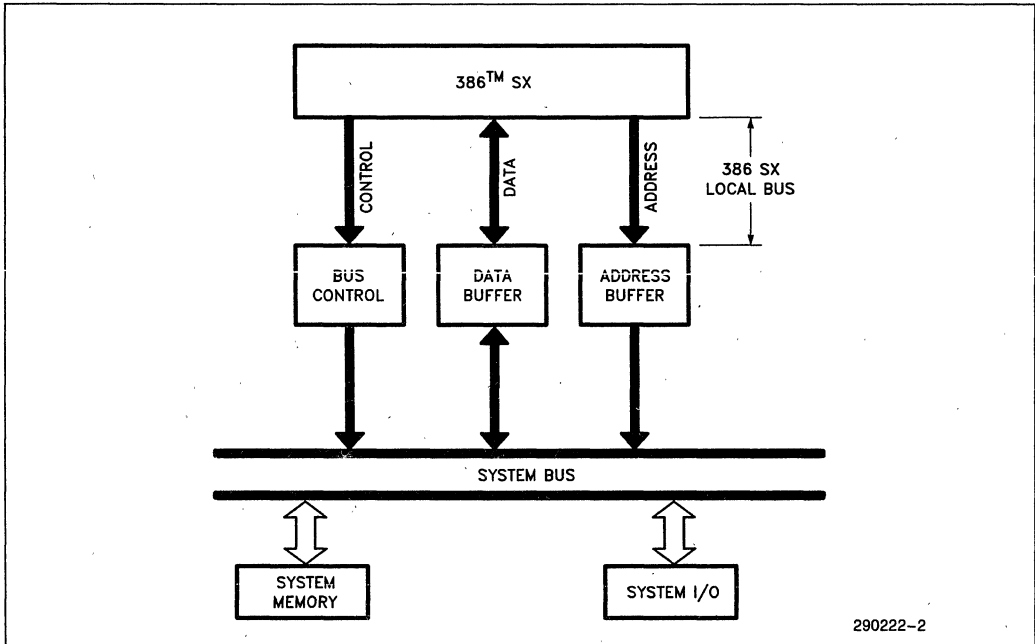


Figure 1-1. 386™ SX System Bus Structure

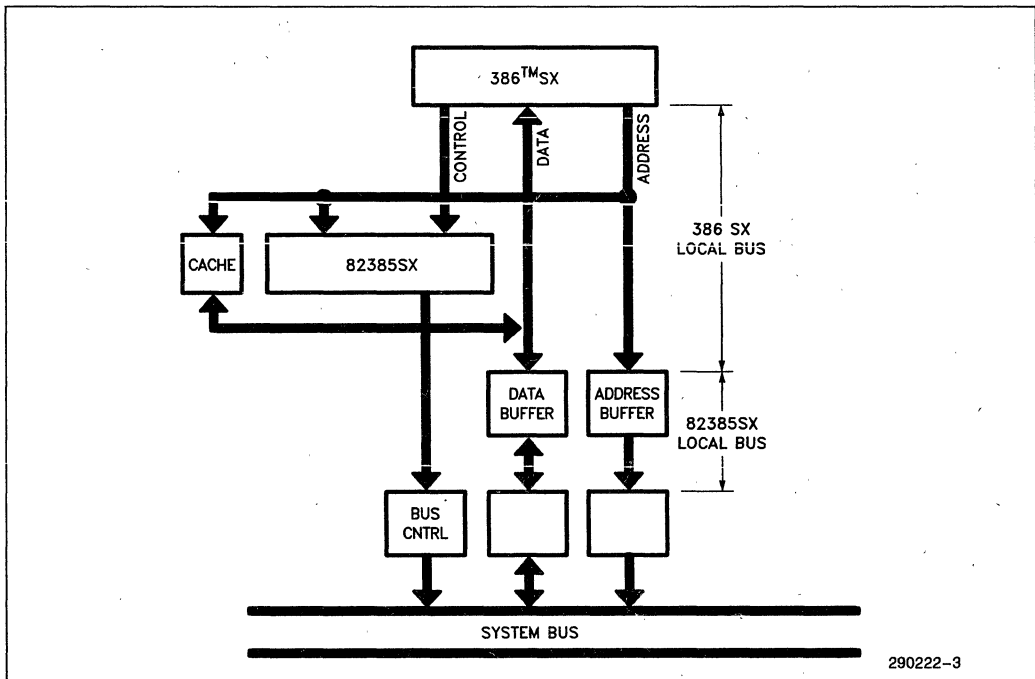
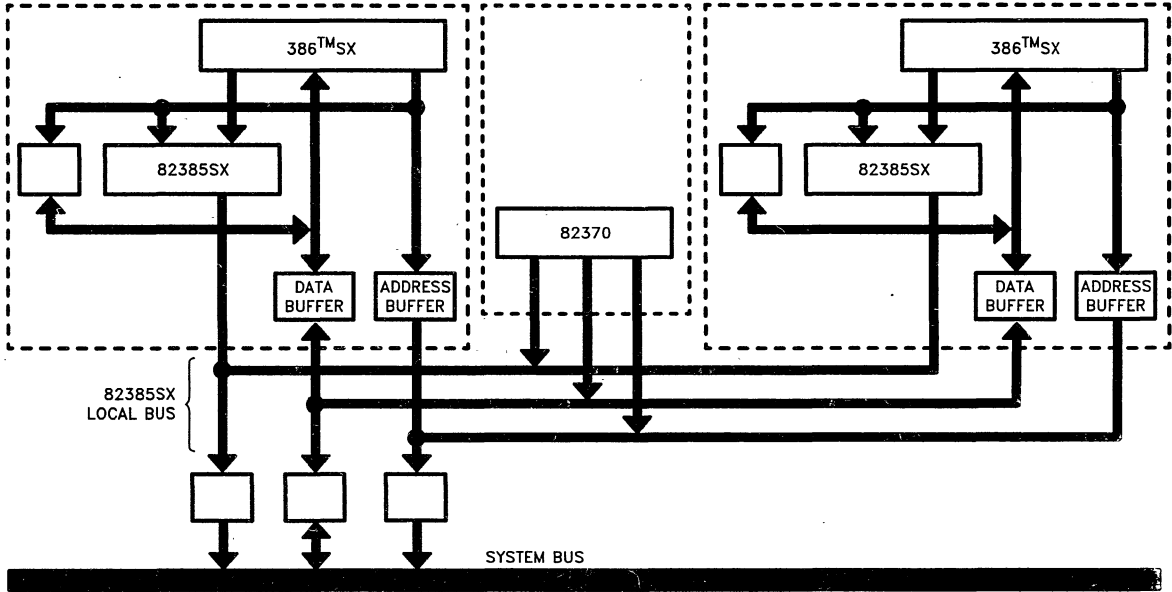


Figure 1-2. 386™ SX and 82385SX System Bus Structure



290222-4

Figure 1-3. Multi-Master/Multi-Cache Environment

1.2.3 MASTER/SLAVE OPERATION

The above 82385SX local bus arbitration discussion is true when the 82385SX is programmed for "Master" mode operation. The user can, however, configure the 82385SX for "Slave" mode operation. (Programming is done via a hardware strap option.) The roles of B HOLD and B HLDA are reversed for an 82385SX in slave mode; B HOLD becomes an output indicating a request to control the bus, and B HLDA becomes an input indicating that a request has been granted. An 82385SX programmed in slave mode drives the 82385SX local bus only when it has requested and subsequently been granted bus control. This allows multiple 386 SX CPU/82385SX subsystems to reside on the same 82385SX local bus (Figure 1-3).

1.2.4 CACHE COHERENCY

Ideally, a cache contains a copy of the most heavily used portions of main memory. To maintain cache "coherency" is to make sure that this local copy is identical to main memory. In a system where multiple masters can access the same memory, there is

always a risk that one master will alter the contents of a memory location that is duplicated in the local cache of another master. (The cache is said to contain "stale" data.) One rather restrictive solution is to not allow cache subsystems to cache shared memory. Another simple solution is to flush the cache anytime another master writes to system memory. However, this can seriously degrade system performance as excessive cache flushing will reduce the hit rate of what may otherwise be a highly efficient cache.

The 82385SX preserves cache coherency via "bus watching" (also called snooping), a technique that neither impacts performance nor restricts memory mapping. An 82385SX that is not currently bus master monitors system bus cycles, and when a write cycle by another master is detected (a snoop), the system address is sampled and used to see if the referenced location is duplicated in the cache. If so (a snoop hit), the corresponding cache entry is invalidated, which will force the 386 SX to fetch the up-to-date data from main memory the next time it accesses this modified location. Figure 1-4 depicts the general form of bus watching.

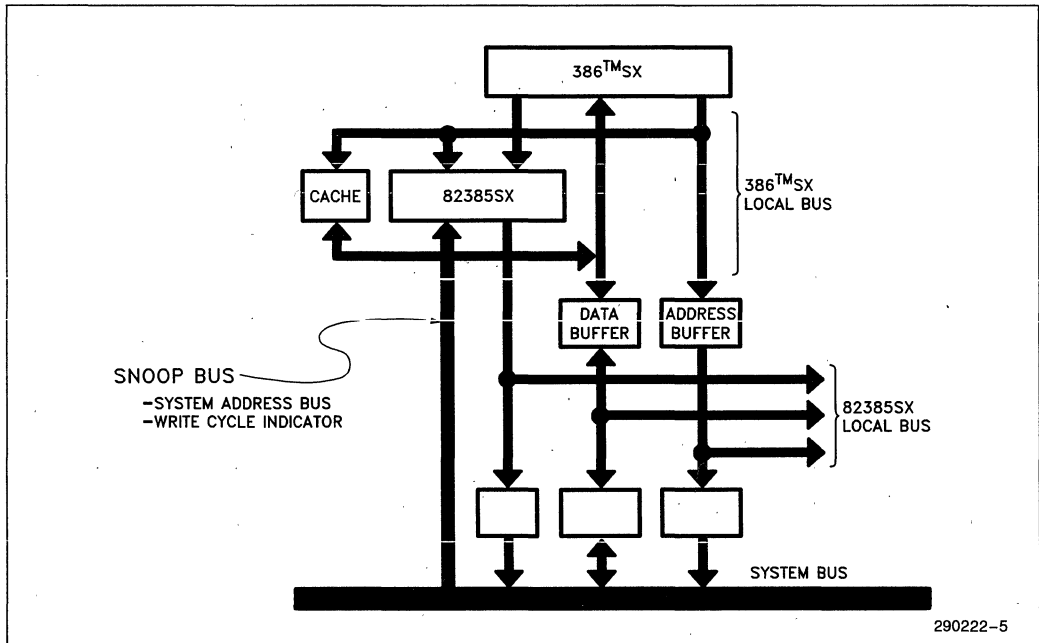


Figure 1-4. 82385SX Bus Watching—Monitor System Bus Write Cycles

1.3 System Overview II: Basic Operation

This discussion is an overview of the basic operation of a 386 SX CPU/82385SX system. Items discussed include the 82385SX's response to all 386 SX cycles, including interrupt acknowledges, halts, and shutdowns. Also discussed are non-cacheable and local accesses.

1.3.1 386™ SX MEMORY CODE AND DATA READ CYCLES

1.3.1.1 Read Hits

When the 386 SX initiates a memory code or data read cycle, the 82385SX compares the high order bits of the 386 SX address bus with the appropriate addresses (tags) stored in its on-chip directory. (The directory structure is described in Section 2.1.1) If the 82385SX determines that the requested data is in the cache, it issues the appropriate control signals that direct the cache to drive the requested data onto the 386 SX data bus, where it is read by the 386 SX. The 82385SX terminates the 386 SX cycle without inserting any wait states.

1.3.1.2 Read Misses

If the 82385SX determines that the requested data is not in the cache, the request is forwarded to the 82385SX local bus and the data retrieved from main memory. As the data returns from main memory, it is directed to the 386 SX and also written into the cache. Concurrently, the 82385SX updates the cache directory such that the next time this particular piece of information is requested by the 386 SX, the 82385SX will find it in the cache and return it with zero wait states.

The basic unit of transfer between main memory and cache memory in a cache subsystem is called the line size. In an 82385SX system, the line size is one 16-bit word. During a read miss, both 82385SX local bus byte enables are active. This insures that the 16-bit entry is written into the cache. (The 386 SX simply ignores what it did not request.) In any other type of 386 SX cycle that is forwarded to the 82385SX local bus, the logic levels of the 386 SX byte enables are duplicated on the 82385SX local bus.

The 82385SX does not actively fetch main memory data independently of the 386 SX. The 82385SX is essentially a passive device which only monitors the address bus and activates control signals. The read miss is the only mechanism by which main memory data is copied into the cache and validated in the cache directory.

In an isolated read miss, the number of wait states seen by the 386 SX is that required by the system memory to respond with data plus the cache comparison cycle (hit/miss decision). The cache system must determine that the cycle is a miss before it can begin the system memory access. However, since misses most often occur consecutively, the 82385SX will begin 386 SX address pipelined cycles to effectively "hide" the comparison cycle beyond the first miss (refer to Section 4.1.3).

The 82385SX can execute a memory access on the 82385SX local bus only if it currently owns the bus. If not, an 82385SX in master mode will run the cycle after the current master releases the bus. An 82385SX in slave mode will issue a hold request, and will run the cycle as soon as the request is acknowledged. (This is true for any read or write cycle that needs to run on the 82385SX local bus.)

1.3.2 386™ SX MEMORY WRITE CYCLES

The 82385SX's "posted write" capability allows the majority of 386 SX memory write cycles to run with zero wait states. The primary memory update policy implemented in a posted write is the traditional cache "write through" technique, which implies that main memory is always updated in any memory write cycle. If the referenced location also happens to reside in the cache (a write hit), the cache is updated as well.

Beyond this, a posted write latches the 386 SX address, data, and cycle definition signals, and the 386 SX local bus is terminated without any wait states, even though the corresponding 82385SX local bus cycle is not yet completed, or perhaps not even started. A posted write is possible because the 82385SX's bus state machine, which is almost identical to the 386 SX bus state machine, is able to run 82385SX local bus cycles independently of the 386 SX. The only time the 386 SX sees write cycle wait states is when a previously latched (posted) write has not yet been completed on the 82385SX local bus or during an I/O write (which is not posted). An 386 SX write can be posted even if the 82385SX does not currently own the 82385SX local bus. In this case, an 82385SX in master mode will run the cycle as soon as the current master releases the bus, and an 82385SX in slave mode will request the bus and run the cycle when the request is acknowledged. The 386 SX is free to continue operating out of its cache (on the 386 SX local bus) during this time.

1.3.3 NON-CACHEABLE CYCLES

Non-cacheable cycles fall into one of two categories: cycles decoded as non-cacheable, and cycles

that are by default non-cacheable according to the 82385SX's design. All non-cacheable cycles are forwarded to the 82385SX local bus. Non-cacheable cycles have no effect on the cache or cache directory.

The 82385SX allows the system designer to define areas of main memory as non-cacheable. The 386 SX address bus is decoded and the decode output is connected to the 82385SX's non-cacheable access (NCA#) input. This decoding is done in the first 386 SX bus state in which the non-cacheable cycle address becomes available. Non-cacheable read cycles resemble cacheable read miss cycles, except that the cache and cache directory are unaffected. NCA# defined non-cacheable writes, like most writes, are posted.

The 82385SX defines certain cycles as non-cacheable without using its non-cacheable access input. These include I/O cycles, interrupt acknowledge cycles, and halt/shutdown cycles. I/O reads and interrupt acknowledge cycles execute as any other non-cacheable read. I/O write cycles are not posted. The 386 SX is not allowed to continue until a ready signal is returned from the system. Halt/Shutdown cycles are posted. During a halt/shutdown condition, the 82385SX local bus duplicates the behavior of the 386 SX, including the ability to recognize and respond to a B HOLD request. (The 82385SX's bus watching mechanism is functional in this condition.)

1.3.4 386™ SX LOCAL BUS CYCLES

386 SX Local Bus Cycles are accesses to resources on the 386 SX local bus other than to the 82385SX itself. The 82385SX simply ignores these accesses: they are neither forwarded to the system nor do they affect the cache. The designer sets aside memory and/or I/O space for local resources by decoding the 386 SX address bus and feeding the decode to the 82385SX's local bus access (LBA#) input. The designer can also decode the 386 SX cycle definition signals to keep specific 386 SX cycles from being forwarded to the system. For example, a multiprocessor design may wish to capture and remedy a 386 SX shutdown locally without having it detected by the rest of the system. Note that in such a design, the local shutdown cycle must be terminated by local bus control logic. The 387 SX Math Coprocessor is considered a 386 SX local bus resource, but it need not be decoded as such by the user since the 82385SX is able to internally recognize 387 SX accesses via the M/IO# and A23 pins.

1.3.5 SUMMARY OF 82385SX RESPONSE TO ALL 386™ SX CYCLES

Table 1-1 summarizes the 82385SX response to all 386 SX bus cycles, as conditioned by whether or not the cycle is decoded as local or non-cacheable. The table describes the impact of each cycle on the cache and on the cache directory, and whether or not the cycle is forwarded to the 82385SX local bus. Whenever the 82385SX local bus is marked "IDLE", it implies that this bus is available to other masters.

1.3.6 BUS WATCHING

As previously discussed, the 82385SX "qualifies" a 386 SX bus cycle in the first bus state in which the address and cycle definition signals of the cycle become available. The cycle is qualified as read or write, cacheable or non-cacheable, etc. Cacheable cycles are further classified as hit or miss according to the results of the cache comparison, which accesses the 82385SX directory and compares the appropriate directory location (tag) to the current 386 SX address. If the cycle turns out to be non-cacheable or a 386 SX local bus access, the hit/miss decision is ignored. The cycle qualification requires one 386 SX state. Since the fastest 386 SX access is two states, the second state can be used for bus watching.

When the 82385SX does not own the system bus, it monitors system bus cycles. If another master writes into main memory, the 82385SX latches the system address and executes a cache look-up to see if the altered main memory location resides in the cache. If so (a snoop hit), the cache entry is marked invalid in the cache directory. Since the directory is at most only being used every other state to qualify 386 SX accesses, snoop look-ups are interleaved between 386 SX local bus look-ups. The cache directory is time multiplexed between the 386 SX address and the latched system address. The result is that all snoops are caught and serviced without slowing down the 386 SX, even when running zero wait state hits on the 386 SX local bus.

1.3.7 CACHE FLUSH

The 82385SX offers a cache flush input. When activated, this signal causes the 82385SX to invalidate all data which had previously been cached. Specifically, all tag valid bits are cleared. (Refer to the 82385SX directory structure in Section 2.1.1.) There-

Table 1-1. 82385SX Response to 386™ SX Cycles

386 SX Bus Cycle Definition					82385SX Response when Decoded as Cacheable			82385SX Response when Decoded as Non-Cacheable			82385SX Response when Decoded as a 386SX Local Bus Access		
M/I/O#	D/C#	W/R#	386 SX Cycle		Cache	Cache Directory	82385SX Local Bus	Cache	Cache Directory	82385SX Local Bus	Cache	Cache Directory	82385SX Local Bus
0	0	0	INT ACK	N/A	—	—	INT ACK	—	—	INT ACK	—	—	IDLE
0	0	1	UNDEFINED	N/A			UNDEFINED			UNDEFINED			IDLE
0	1	0	I/O READ	N/A	—	—	I/O READ	—	—	I/O READ	—	—	IDLE
0	1	1	I/O WRITE	N/A	—	—	I/O WRITE	—	—	I/O WRITE	—	—	IDLE
1	0	0	MEM CODE READ	HIT	CACHE READ	—	IDLE	—	—	MEM CODE READ	—	—	IDLE
				MISS	CACHE WRITE	DATA VALIDATION	MEM CODE READ						
1	0	1	HALT/SHUTDOWN	N/A	—	—	HALT/SHUTDOWN	—	—	HALT/SHUTDOWN	—	—	IDLE
1	1	0	MEM DATA READ	HIT	CACHE READ	—	IDLE	—	—	MEM DATA READ	—	—	IDLE
				MISS	CACHE WRITE	DATA VALIDATION	MEM DATA READ						
1	1	1	MEM DATA WRITE	HIT	CACHE WRITE	—	MEM DATA WRITE	—	—	MEM DATA WRITE	—	—	IDLE
				MISS	—	—	MEM DATA WRITE						

NOTES:

- A dash (—) indicates that the cache and cache directory are unaffected. This table does not reflect how an access affects the LRU bit.
- An "IDLE" 82385SX Local Bus implies that this bus is available to other masters.
- The 82385SX's response to 387™ SX accesses is the same as when decoded as a 386 SX Local Bus Access.
- The only other operations that affect the cache directory are:
 1. RESET or Cache Flush—all tag valid bits cleared.
 2. Snoop Hit—corresponding line valid bit cleared.

fore, the cache is empty and subsequent cycles are misses until the 386 SX begins repeating the new accesses (hits). The primary use of the FLUSH input is for diagnostics and multi-processor support.

NOTE:

The use of this pin as a coherency mechanism may impact software transparency.

2.0 82385SX CACHE ORGANIZATION

The 82385SX supports two cache organizations: a simple direct mapped organization and a slightly more complex, higher performance two way set associative organization. The choice is made by strapping an 82385SX input (2W/D#) either high or low. This chapter describes the structure and operation of both organizations.

2.1 Direct Mapped Cache

2.1.1 DIRECT MAPPED CACHE STRUCTURE AND TERMINOLOGY

Figure 2-1 depicts the relationship between the 82385SX's internal cache directory, the external cache memory, and the 386 SX's physical address space. The 386 SX address space can conceptually

be thought of as cache "pages" each being 8K words (16 Kbytes) deep. The page size matches the cache size. The cache can be further divided into 1024 (0 thru 1023) sets of eight words (8 x 16 bits). Each 16-bit word is called a "line". The unit of transfer between the main memory and cache is one line.

Each block in the external cache has an associated 19-bit entry in the 82385SX's internal cache directory. This entry has three components: a 10-bit "tag", a "tag valid" bit, and eight "line valid" bits. The tag acts as a main memory page number (10 tag bits support 2^{10} pages). For example, if line 9 of page 2 currently resides in the cache, then a binary 2 is stored in the Set 1 tag field. (For any 82385SX direct mapped cache page in main memory, Set 0 consists of lines 0-7, Set 1 consists of lines 8-15, etc. Line 9 is shaded in Figure 2-1.) An important characteristic of a direct mapped cache is that line 9 of any page can only reside in line 9 of the cache. All identical page offsets map to a single cache location.

The data in a cache set is considered valid or invalid depending on the status of its tag valid bit. If clear, the entire set is considered invalid. If true, an individual line within the set is considered valid or invalid depending on the status of its line valid bit.

The 82385SX sees the 386 SX address bus (A1-A23) as partitioned into three fields: a 10-bit "tag"

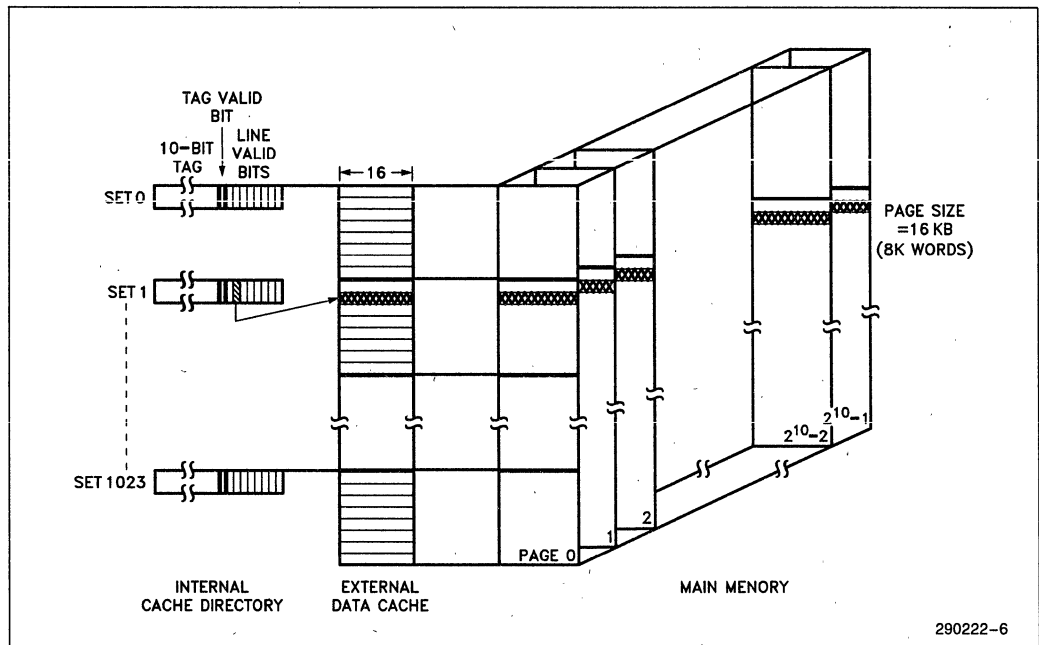


Figure 2-1. Direct Mapped Cache Organization

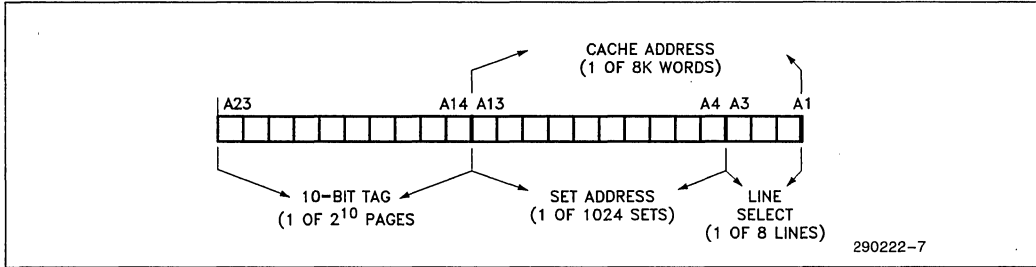


Figure 2-2. 386™ SX Address Bus Bit Fields—Direct Mapped Organization

field (A14–A23), a 10-bit “set address” field (A4–A13), and a 3-bit “line select” field (A1–A3). (See Figure 2-2.) The lower 13 address bits (A1–A13) also serve as the “cache address” which directly selects one of 8K words in the external cache.

stored tag, the tag valid bit is set, the appropriate line valid bit is set, and the other seven line valid bits are cleared. Subsequent tag hits with line misses will only set the appropriate line valid bit. (Any data associated with the previous tag is no longer considered resident in the cache.)

2.1.2 DIRECT MAPPED CACHE OPERATION

The following is a description of the interaction between the 386 SX, cache, and cache directory.

2.1.2.1 Read Hits

When the 386 SX initiates a memory read cycle, the 82385SX uses the 10-bit set address to select one of 1024 directory entries, and the 3-bit line select field to select one of eight line valid bits within the entry. The 13-bit cache address selects the corresponding word in the cache. The 82385SX compares the 10-bit tag field (A14–A23 of the 386 SX access) with the tag stored in the selected directory entry. If the tag and upper address bits match, and if both the tag and appropriate line valid bits are set, the result is a hit, and the 82385SX directs the cache to drive the selected word onto the 386 SX data bus. A read hit does not alter the contents of the cache or directory.

2.1.2.2 Read Misses

A read miss can occur in two ways. The first is known as a “line” miss, and occurs when the tag and upper address bits match and the tag valid bit is set, but the line valid bit is clear. The second is called a “tag” miss, and occurs when either the tag and upper address bits do not match, or the tag valid bit is clear. (The line valid bit is a “don’t care” in a tag miss.) In both cases, the 82385SX forwards the 386 SX reference to the system, and as the returning data is fed to the 386 SX, it is written into the cache and validated in the cache directory.

In a line miss, the incoming data is validated simply by setting the previously clear line valid bit. In a tag miss, the upper address bits overwrite the previously

2.1.2.3 Other Operations That Affect the Cache and Cache Directory

The other operations that affect the cache and/or directory are write hits, snoop hits, cache flushes, and 82385SX resets. In a write hit, the cache is updated along with main memory, but the directory is unaffected. In a snoop hit, the cache is unaffected, but the affected line is invalidated by clearing its line valid bit in the directory. Both an 82385SX reset and cache flush clear all tag valid bits.

When a 386 SX CPU/82385SX system “wakes up” upon reset, all tag valid bits are clear. At this point, a read miss is the only mechanism by which main memory data is copied into the cache and validated in the cache directory. Assume an early 386 SX code access seeks (for the first time) line 9 of page 2. Since the tag valid bit is clear, the access is a tag miss, and the data is fetched from main memory. Upon return, the data is fed to the 386 SX and simultaneously written into line 9 of the cache. The set directory entry is updated to show this line as valid. Specifically, the tag and appropriate line valid bits are set, the remaining seven line valid bits cleared, and binary 2 written into the tag. Since code is sequential in nature, the 386 SX will likely next want line 10 of page 2, then line 11, and so on. If the 386 SX sequentially fetches the next six lines, these fetches will be line misses, and as each is fetched from main memory and written into the cache, its corresponding line valid bit is set. This is the basic flow of events that fills the cache with valid data. Only after a piece of data has been copied into the cache and validated can it be accessed in a zero wait state read hit. Also, a cache entry must have been validated before it can be subsequently altered by a write hit, or invalidated by a snoop hit.

An extreme example of “thrashing” is if line 9 of page two is an instruction to jump to line 9 of page one, which is an instruction to jump back to line 9 of page two. Thrashing results from the direct mapped cache characteristic that all identical page offsets map to a single cache location. In this example, the page one access overwrites the cached page two data, and the page two access overwrites the cached page one data. As long as the code jumps back and forth the hit rate is zero. This is of course an extreme case. The effect of thrashing is that a direct mapped cache exhibits a slightly reduced overall hit rate as compared to a set associative cache of the same size.

2.2 Two Way Set Associative Cache

2.2.1 TWO WAY SET ASSOCIATIVE CACHE STRUCTURE AND TERMINOLOGY

Figure 2-3 illustrates the relationship between the directory, cache, and 386 SX address space. Whereas the direct mapped cache is organized as one bank of 8K words, the two way set associative cache is organized as two banks (A and B) of 4K words each. The page size is halved, and the number of pages doubled. (Note the extra tag bit.) The cache now has 512 sets in each bank. (Two banks times 512 sets gives a total of 1024. The structure can be thought of as two half-sized direct mapped caches in parallel.) The performance advantage over a direct mapped cache is that all identical page offsets map to two cache locations instead of one, reducing the potential for thrashing. The 82385SX's partitioning of the 386 SX address bus is depicted in Figure 2-4.

2.2.2 LRU REPLACEMENT ALGORITHM

The two way set associative directory has an additional feature: the “least recently used” or LRU bit. In the event of a read miss, either bank A or bank B will be updated with new data. The LRU bit flags the candidate for replacement. Statistically, of two blocks of data, the block most recently used is the block most likely to be needed again in the near future. By flagging the least recently used block, the 82385SX ensures that the cache block replaced is the least likely to have data needed by the CPU.

2.2.3 TWO WAY SET ASSOCIATIVE CACHE OPERATION

2.2.3.1 Read Hits

When the 386 SX initiates a memory read cycle, the 82385SX uses the 9-bit set address to select one of

512 sets. The two tags of this set are simultaneously compared with A13–A23, both tag valid bits checked, and both appropriate line valid bits checked. If either comparison produces a hit, the corresponding cache bank is directed to drive the selected word onto the 386 SX data bus. (Note that both banks will never concurrently cache the same main memory location.) If the requested data resides in bank A, the LRU bit is pointed toward B. If B produces the hit, the LRU bit is pointed toward A.

2.2.3.2 Read Misses

As in direct mapped operation, a read miss can be either a line or tag miss. Let's start with a tag miss example. Assume the 386 SX seeks line 9 of page 2, and that neither the A or B directory produces a tag match. Assume also, as indicated in Figure 2-3, that the LRU bit points to A. As the data returns from main memory, it is loaded into offset 9 of bank A. Concurrently, this data is validated by updating the set 1 directory entry for bank A. Specifically, the upper address bits overwrite the previous tag, the tag valid bit is set, the appropriate line valid bit is set, and the other seven line valid bits cleared. Since this data is the most recently used, the LRU bit is turned toward B. No change to bank B occurs.

If the next 386 SX request is line 10 of page two, the result will be a line miss. As the data returns from main memory, it will be written into offset 10 of bank A (tag hit/line miss in bank A), and the appropriate line valid bit will be set. A line miss in one bank will cause the LRU bit to point to the other bank. In this example, however, the LRU bit has already been turned toward B.

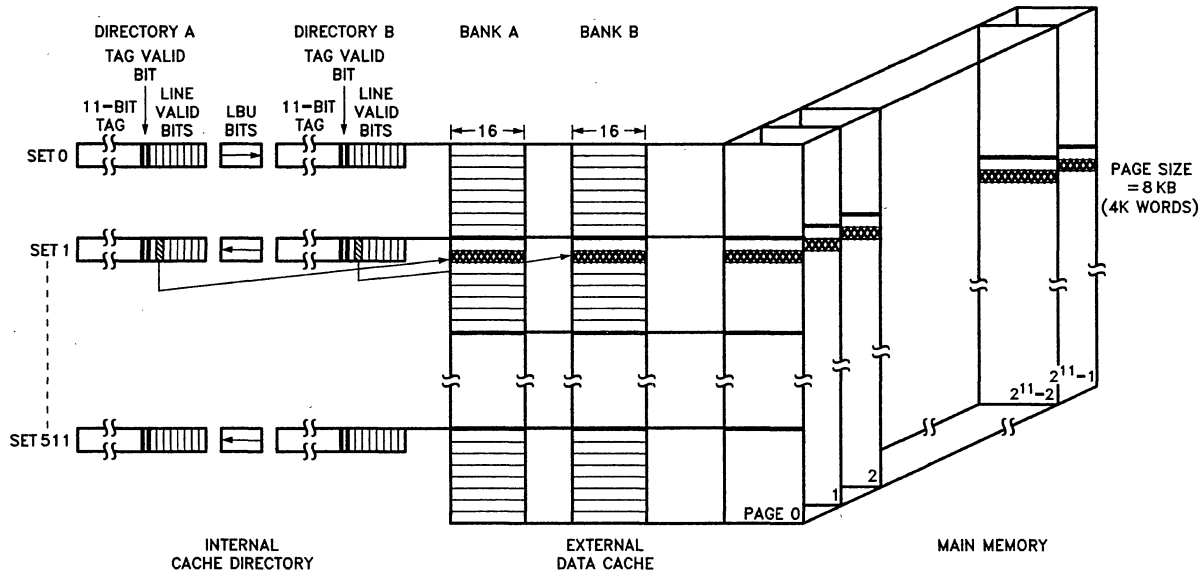
2.2.3.3 Other Operations That Affect the Cache and Cache Directory

Other operations that affect the cache and cache directory are write hits, snoop hits, cache flushes, and 82385SX resets. A write hit updates the cache along with main memory. If directory A detects the hit, bank A is updated. If directory B detects the hit, bank B is updated. If one bank is updated, the LRU bit is pointed towards the other.

If a snoop hit invalidates an entry, for example, in cache bank A, the corresponding LRU bit is pointed toward A. This insures that invalid data is the prime candidate for replacement in a read miss. Finally, resets and flushes behave just as they do in a direct mapped cache, clearing all tag valid bits.

3.0 82385SX PIN DESCRIPTION

The 82385SX creates the 82385SX local bus, which is a functional 386 SX interface. To facilitate under-



290222-8

Figure 2-3. Two-Way Set Associative Cache Organization

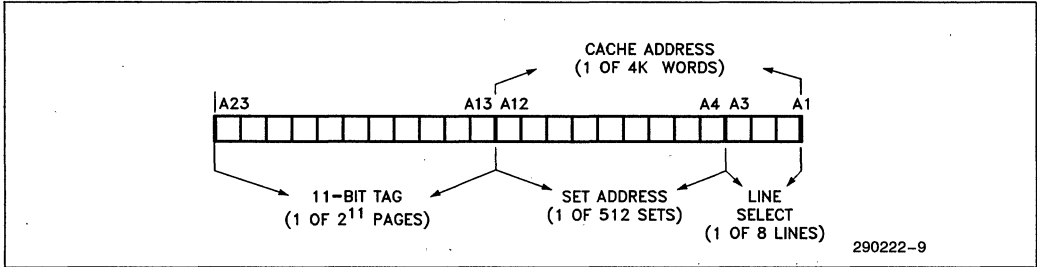


Figure 2-4. 386™ SX Address Bus Bit Fields—Two-Way Set Associative Organization

standing, 82385SX local bus signals go by the same name as their 386 SX equivalents, except that they are preceded by the letter "B". The 82385SX local bus equivalent to ADS# is BADS#, the equivalent to NA# is BNA#, etc. This convention applies to bus states as well. For example, BT1P is the 82385SX local bus state equivalent to the 386 SX T1P state.

82385SX, like the 386 SX, divides CLK2 by two to generate an internal "phase indication" clock. (See Figure 3-1.) The CLK2 period whose rising edge drives the internal clock low is called PHI1, and the CLK2 period that drives the internal clock high is called PHI2. A PHI1-PHI2 combination (in that order) is known as a "T" state, and is the basis for 386 SX bus cycles.

3.1 386™ SX CPU/82385SX Interface Signals

These signals form the direct interface between the 386 SX and the 82385SX.

3.1.1 386™ SX CPU/82385SX Clock (CLK2)

CLK2 provides the fundamental timing for a 386 SX CPU/82385SX system, and is driven by the same source that drives the 386 SX CLK2 input. The

3.1.2 386™ SX CPU/82385SX RESET (RESET)

This input resets the 82385SX, bringing it to an initial known state, and is driven by the same source that drives the 386 SX RESET input. A reset effectively flushes the cache by clearing all cache directory tag valid bits. The falling edge of RESET is synchronized to CLK2, and used by the 82385SX to properly establish the phase of its internal clock. (See Figure 3-2.) Specifically, the second internal phase following the falling edge of RESET is PHI2.

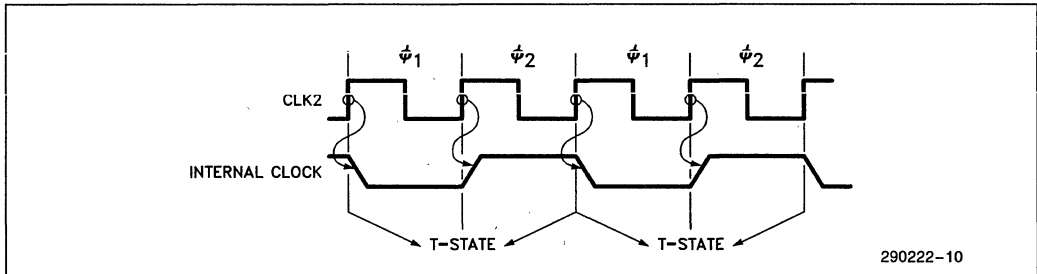


Figure 3-1. CLK2 and Internal Clock

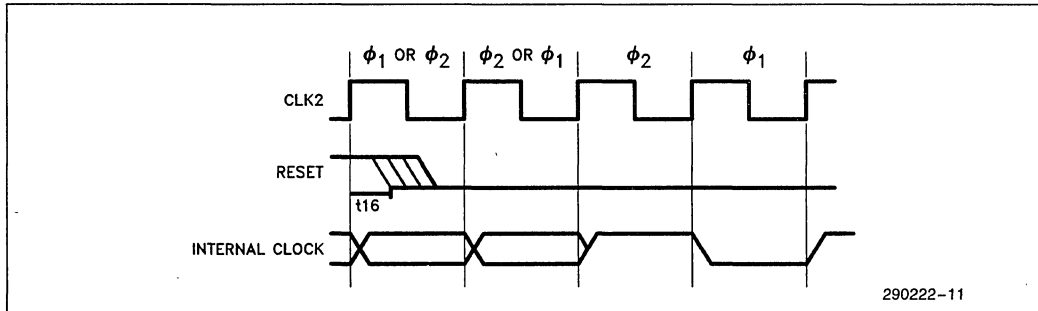


Figure 3-2. Reset/Internal Phase Relationship

3.1.3 386™ SX CPU/82385SX ADDRESS BUS (A1–A23), BYTE ENABLES (BHE#, BLE#), AND CYCLE DEFINITION SIGNALS (M/IO#, D/C#, W/R#, LOCK#)

The 82385SX directly connects to these 386 SX outputs. The 386 SX address bus is used in the cache directory comparison to see if data referenced by 386 SX resides in the cache, and the byte enables inform the 82385SX as to which portions of the data bus are involved in a 386 SX cycle. The cycle definition signals are decoded by the 82385SX to determine the type of cycle the 386 SX is executing.

3.1.4 386™ SX CPU/82385SX ADDRESS STATUS (ADS#) AND READY INPUT (READYI#)

ADS#, a 386 SX output, tells the 82385SX that new address and cycle definition information is available. READYI#, an input to both the 386 SX (via the 386 SX READY# input pin) and 82385SX, indicates the completion of a 386 SX bus cycle. ADS# and READYI# are used to track the 386 SX bus state.

3.1.5 386™ SX NEXT ADDRESS REQUEST (NA#)

This 82385SX output controls 386 SX pipelining. It can be tied directly to the 386 SX NA# input, or it can be logically “AND”ed with other 386 SX local bus next address requests.

3.1.6 READY OUTPUT (READYO#) AND BUS READY ENABLE (BRDYEN#)

The 82385SX directly terminates all but two types of 386 SX bus cycles with its READYO# output. 386 SX local bus cycles must be terminated by the local device being accessed. This includes devices decoded using the 82385SX LBA# signal and 387 accesses. The other cycles not directly terminated by the 82385SX are 82385SX local bus reads, spe-

cifically cache read misses and non-cacheable reads. (Recall that the 82385SX forwards and runs such cycles on the 82385SX bus.) In these cycles the signal that terminates the 82385SX local bus access is BREADY# which is gated through to the 386 SX local bus such that the 386 SX and 82385SX local bus cycles are concurrently terminated. BRDYEN# is used to gate the BREADY# signal to the 386 SX.

3.2 Cache Control Signals

These 82385SX outputs control the external 16 KB cache data memory.

3.2.1 CACHE ADDRESS LATCH ENABLE (CALEN)

This signal controls the latch (typically an F or AS series 74373) that resides between the low order 386 SX address bits and the cache SRAM address inputs. (The outputs of this latch are the “cache address” described in the previous chapter.) When CALEN is high the latch is transparent. The falling edge of CALEN latches the current inputs which remain applied to the cache data memory until CALEN returns to an active high state.

3.2.2 CACHE TRANSMIT/RECEIVE (CT/R#)

This signal defines the direction of an optional data transceiver (typically an F or AS series 74245) between the cache and 386 SX data bus. When high, the transceiver is pointed towards the 386 SX local data bus (the SRAMs are output enabled). When low, the transceiver points towards the cache data memory. A transceiver is required if the cache is designed with SRAMs that lack an output enable control. A transceiver may also be desirable in a system that has a heavily loaded 386 SX local data bus. These devices are not necessary when using SRAMs which incorporate an output enable.

3.2.3 CACHE CHIP SELECTS (CS0#, CS1#)

These active low signals tie to the cache SRAM chip selects, and individually enable both bytes of the 16-bit wide cache. CS0# enables D0–D7 and CS1# enables D8–D15. During read hits, both bytes are enabled regardless of whether or not the 386 SX byte enables are active. (The 386 SX ignores what it did not request.) Also, both cache bytes are enabled in a read miss so as to update the cache with a complete line (word). In a write hit, only the cache bytes that correspond to active byte enables are selected. This prevents cache data from being corrupted in a partial word write.

3.2.4 CACHE OUTPUT ENABLES (COEA#, COEB#) AND WRITE ENABLES (CWEA#, CWEB#)

COEA# and COEB# are active low signals which tie to the cache SRAM or Transceiver output enables and respectively enable cache bank A or B. The state of DEFOE# (define cache output enable), an 82385SX configuration input, determines the functional definition of COEA# and COEB#.

If DEFOE# = V_{IL} , in a two-way set associative cache, either COEA# or COEB# is active during read hit cycles only, depending on which bank is selected. In a direct mapped cache, both are activated during read hits, so the designer is free to use either one. This COEx# definition best suits cache SRAMs with output enables.

If DEFOE# = V_{IH} , COEx# is active during a read hit, read miss (cache update) and write hit cycles only. This COEx# definition best suits cache SRAMs without output enables. In such systems, transceivers are needed and their output enables must be active for writing, as well as reading, the cache SRAMs.

CWEA# and CWEB# are active low signals which tie to the cache SRAM write enables, and respectively enable cache bank A or B to receive data from the 386 SX data bus (386 SX write hit or read miss update). In a two-way set associative cache, one or the other is enabled in a read miss or write hit. In a direct mapped cache, both are activated, so the designer is free to use either one.

The various cache configurations supported by the 82385SX are described in Section 4.2.1.

3.3 386™ SX Local Bus Decode Inputs

These 82385SX inputs are generated by decoding the 386 SX address and cycle definition lines. These

active low inputs are sampled at the end of the first state in which the address of a new 386 SX cycle becomes available. (T1 or first T2P.)

3.3.1 386™ SX LOCAL BUS ACCESS (LBA#)

This input identifies a 386 SX access as directed to a resource (other than the cache) on the 386 SX local bus. (The 387 SX Math Coprocessor is considered a 386 SX local bus resource, but LBA# need not be generated as the 82385SX internally decodes 387 SX accesses.) The 82385SX simply ignores these cycles. They are neither forwarded to the system nor do they affect the cache or cache directory. Note that LBA# has priority over all other types of cycles. If LBA# is asserted, the cycle is interpreted as a 386 SX local bus access, regardless of the cycle type or status of NCA#. This allows any 386 SX cycle (memory, I/O, interrupt acknowledge, etc.) to be kept on the 386 SX local bus if desired.

3.3.2 NON-CACHEABLE ACCESS (NCA#)

This active low input identifies a 386 SX cycle as non-cacheable. The 82385SX forwards non-cacheable cycles to the 82385SX local bus and runs them. The cache and cache directory are unaffected.

NCA# allows a designer to set aside a portion of main memory as non-cacheable. Potential applications include memory-mapped I/O and systems where multiple masters access dual ported memory via different busses. Another possibility makes use of the 386 SX D/C# output. The 82385SX by default implements a unified code and data cache, but driving NCA# directly by D/C# creates a data only cache. If D/C# is inverted first, the result is a code only cache.

3.4 82385SX Local Bus Interface Signals

The 82385SX presents an "386 SX-like" front end to the system, and the signals discussed in this section are 82385SX local bus equivalents to actual 386 SX signals. These signals are named with respect to their 386 SX counterparts, but with the letter "B" appended to the front.

Note that the 82385SX itself does not have equivalent output signals to the 386 SX data bus (D0–D15) address bus (A1–A23), and cycle definition signals (M/IO#, D/C#, W/R#). The 82385SX data bus (BD0–BD15) is actually the system side of a latching transceiver, and the 82385SX address bus and cycle definition signals (BA1–BA23, BM/IO#, BD/C#,

BW/R#) are the outputs of an edge-triggered latch. The signals that control this data transceiver and address latch are discussed in Section 3.5.

3.4.1 82385SX BUS BYTE ENABLES (BBHE#, BBLE#)

BBHE# and BBLE# are the 82385SX local bus equivalents to the 386 SX byte enables. In a cache read miss, the 82385SX drives both signals low, regardless of whether or not the 386 SX byte enables are active. This insures that a complete line (word) is fetched from main memory for the cache update. In all other 82385SX local bus cycles, the 82385SX duplicates the logic levels of the 386 SX byte enables. The 82385SX tri-states these outputs when it is not the current bus master.

3.4.2 82385SX BUS LOCK (BLOCK#)

BLOCK# is the 82385SX local bus equivalent to the 386 SX LOCK# output, and distinguishes between locked and unlocked cycles. When the 386 SX runs a locked sequence of cycles (and LBA# is negated), the 82385SX forwards and runs the sequence on the 82385SX local bus, regardless of whether any locations referenced in the sequence reside in the cache. A read hit will be run as if it is a read miss, but a write hit will update the cache as well as being completed to system memory. In keeping with 386 SX behavior, the 82385SX does not allow another master to interrupt the sequence. BLOCK# is tri-stated when the 82385SX is not the current bus master.

3.4.3 82385SX BUS ADDRESS STATUS (BADS#)

BADS# is the 82385SX local bus equivalent of ADS#, and indicates that a valid address (BA1–BA23, BBHE#, BBLE#) and cycle definition (BM/IO#, BW/R#, BD/C#) are available. It is asserted in BT1 and BT2P states, and is tri-stated when the 82385SX does not own the bus.

3.4.4 82385SX BUS READY INPUT (BREADY#)

82385SX local bus cycles are terminated by BREADY#, just as 386 SX cycles are terminated by the 386 SX READY# input. In 82385SX local bus read cycles, BREADY# is gated by BRDYEN# onto the 386 SX local bus, such that it terminates both the 386 SX and 82385SX local bus cycles.

3.4.5 82385SX BUS NEXT ADDRESS REQUEST (BNA#)

BNA# is the 82385SX local bus equivalent to the 386 SX NA# input, and indicates that the system is

prepared to accept a pipelined address and cycle definition. If BNA# is asserted and the new cycle information is available, the 82385SX begins a pipelined cycle on the 82385SX local bus.

3.5 82385SX Bus Data Transceiver and Address Latch Control Signals

The 82385SX data bus is the system side of a latching transceiver (typically for F or AS series 74646), and the 82385SX address bus and cycle definition signals are the outputs of an edge-triggered latch (F or AS series 74374). The following is a discussion of the 82385SX outputs that control these devices. An important characteristic of these signals and the devices they control is that they ensure that BD0–BD15, BA1–BA23, BM/IO#, BD/C# and BW/R# reproduce the functionality and timing behavior of their 386 SX equivalents.

3.5.1 LOCAL DATA STROBE (LDSTB), DATA OUTPUT ENABLE (DOE#), AND BUS TRANSMIT/RECEIVE (BT/R#)

These signals control the latching data transceiver. BT/R# defines the transceiver direction. When high, the transceiver drives the 82385SX data bus in write cycles. When low, the transceiver drives the 386 SX data bus in 82385SX local bus read cycles. DOE# enables the transceiver outputs.

The rising edge of LDSTB latches the 386 SX data bus in all write cycles. The interaction of this signal and the latching transceiver is used to perform the 82385SX's posted write capability.

3.5.2 BUS ADDRESS CLOCK PULSE (BACP) AND BUS ADDRESS OUTPUT ENABLE (BAOE#)

These signals control the latch that drives BA1–BA23, BM/IO#, BW/R#, and BD/C#. In any 386 SX cycle that is forwarded to the 82385SX local bus, the rising edge of BACP latches the 386 SX address and cycle definition signals. BAOE# enables the latch outputs when the 82385SX is the current bus master and disables them otherwise.

3.6 Status and Control Signals

3.6.1 CACHE MISS INDICATION (MISS#)

This output accompanies cacheable read and write miss cycles. This signal transitions to its active low state when the 82385SX determines that a cacheable 386 SX access is a miss. Its timing behavior

follows that of the 82385SX local bus cycle definition signals (BM/IO#, BD/C#, BW/R#) so that it becomes available with BADS# in BT1 or the first BT2P. MISS# is floated when the 82385SX does not own the bus, such that multiple 82385SX's can share the same node in multi-cache systems. (As discussed in Chapter 7, this signal also serves a reserved function in testing the 82385SX.)

3.6.2 WRITE BUFFER STATUS (WBS)

The latching data transceiver is also known as the "posted write buffer". WBS indicates that this buffer contains data that has not yet been written to the system even though the 386 SX may have begun its next cycle. It is activated when 386 SX data is latched, and deactivated when the corresponding 82385SX local bus write cycle is completed (BREADY#). (As discussed in Chapter 7, this signal also serves a reserved function in testing the 82385SX.)

WBS can serve several functions. In multi-processor applications, it can act as a coherency mechanism by informing a bus arbiter that it should let a write cycle run on the system bus so that main memory has the latest data. If any other 82385SX cache subsystems are on the bus, they will monitor the cycle via their bus watching mechanisms. Any 82385SX that detects a snoop hit will invalidate the corresponding entry in its local cache.

3.6.3 CACHE FLUSH (FLUSH)

When activated, this signal causes the 82385SX to clear all of its directory tag valid bits, effectively flushing the cache. (As discussed in Chapter 7, this signal also serves a reserved function in testing the 82385SX.) The primary use of the FLUSH input is for diagnostics and multi-processor support. The use of this pin as a coherency mechanism may impact software transparency.

The FLUSH input must be held active for at least 4 CLK (8 CLK2) cycles to complete the flush sequence. If FLUSH is still active after 4 CLK cycles, any accesses to the cache will be misses and the cache will not be updated (since FLUSH is active).

3.7 Bus Arbitration Signals (BHOLD and BHLDA)

In master mode, BHOLD is an input that indicates a request by a slave device for bus ownership. The

82385SX acknowledges this request via its BHLDA output. (These signals function identically to the 386 SX HOLD and HLDA signals.)

The roles of BHOLD and BHLDA are reversed for an 82385SX in slave mode. BHOLD is now an output indicating a request for bus ownership, and BHLDA an input indicating that the request has been granted.

3.8 Coherency (Bus Watching) Support Signals (SA1-SA23, SSTB#, SEN)

These signals form the 82385SX's bus watching interface. The Snoop Address Bus (SA1-SA23) connects to the system address lines if masters reside at both the system and 82385SX local bus levels, or the 82385SX local bus address lines if masters reside only at the 82385SX local bus level. Snoop Strobe (SSTB#) indicates that a valid address is on the snoop address inputs. Snoop Enable (SEN) indicates that the cycle is a write. In a system with masters only at the 82385SX local bus level, SA1-SA23, SSTB#, and SEN can be driven respectively by BA1-BA23, BADS#, and BW/R# without any support circuitry.

3.9 Configuration Inputs (2W/D#, M/S#, DEFOE#)

These signals select the configurations supported by the 82385SX. They are hardware strap options and must not be changed dynamically. 2W/D# (2-Way/Direct Mapped Select) selects a two-way set associative cache when tied high, or a direct mapped cache when tied low. M/S# (Master/Slave Select) chooses between master mode (M/S# high) and slave mode (M/S# low). DEFOE# defines the functionality of the 82385SX cache output enables (COEA# and COEB#). DEFOE# allows the 82385SX to interface to SRAMs with output enables (DEFOE# low) or to SRAMs requiring transceivers (DEFOE# high).

3.10 Reserved Pins (RES)

Some pins on the 82385SX are reserved for internal testing and future cache features. To assure compatibility and functionality, these reserved pins must be configured as shown in Table 3.10.1.

Table 3.10.1. Reserved Pin Connections

PGA Pin Location	PQFP Pin Location	Logic Level
A12	1	High
A13	131	High
B10	7	High
B11	3	High
B12	132	High
C10	4	High
C11	2	High
G13	117	High
H12	110	High
J3	60	High
J14	109	High
K1	58	High
K2	59	High
K3	62	High
L1	61	High
L2	63	High
L3	64	High
L12	100	No Connect
L14	102	High
M13	101	No Connect
N6	75	No Connect
P5	76	No Connect

4.0 386 SX LOCAL BUS INTERFACE

The following is a detailed description of how the 82385SX interfaces to the 386 SX and to 386 SX local bus resources. Items specifically addressed are the interfaces to the 386 SX, the cache SRAMs, and the 387 SX Math Coprocessor.

The many timing diagrams in this and the next chapter provide insight into the dual pipelined bus structure of a 386 SX CPU/82385SX system. It's important to realize, however, that one need not know every possible cycle combination to use the 82385SX. The interface is simple, and the dual bus operation invisible to the 386 SX and system. To facilitate discussion of the timing diagrams, several conventions have been adopted. Refer to Figure 4-2A, and note that 386 SX bus cycles, 386 SX bus states, and 82385SX bus states are identified along the top. All states can be identified by the "frame numbers" along the bottom. The cycles in Figure 4-2A include a cache read hit (CRDH), a cache read miss (CRDM), and a write (WT). WT represents any write, cacheable or not. When necessary to distinguish cacheable writes, a write hit goes by CWTH and a write miss by CWTM. Non-cacheable system reads go by SBRD. Also, it is assumed that system bus pipelining occurs even though the BNA# signal is not shown. When the system pipeline begins is a function of the system bus controller.

386 SX bus cycles can be tracked by ADS# and READYI#, and 82385SX cycles by BADS# and BREADY#. These four signals are thus a natural choice to help track parallel bus activity. Note in the timing diagrams that 386 SX cycles are numbered using ADS# and READYI#, and 82385SX cycles using BADS# and BREADY#. For example, when the address of the first 386 SX cycle becomes available, the corresponding assertion of ADS# is marked "1", and the READYI# pulse that terminates the cycle is marked "1" as well. Whenever a 386 SX cycle is forwarded to the system, its number is forwarded as well so that the corresponding 82385SX bus cycle can be tracked by BADS# and BREADY#.

The "N" value in the timing diagrams is the assumed number of main memory wait states inserted in a non-pipelined 82385SX bus cycle. For example, a non-pipelined access to N=2 memory requires a total of four bus states, while a pipelined access requires three. (The pipeline advantage effectively hides one main memory wait state.)

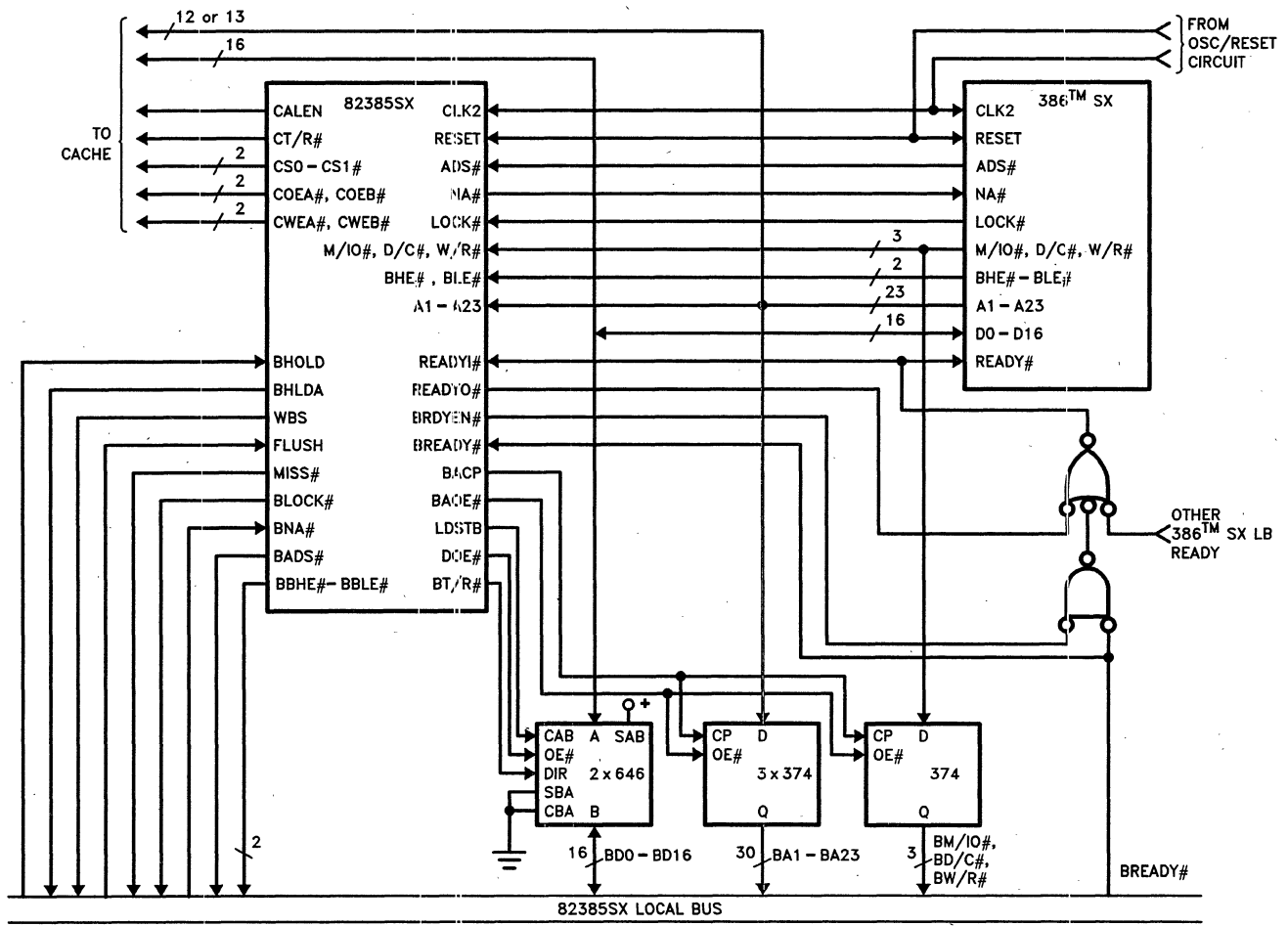
4.1 Processor Interface

This section presents the 386 SX CPU/82385SX hardware interface and discusses the interaction and timing of this interface. Also addressed is how to decode the 386 SX address bus to generate the 82385SX inputs LBA# and NCA#. (Recall that LBA# allows memory and/or I/O space to be set aside for 386 SX local bus resources; and NCA# allows system memory to be set aside as non-cacheable.)

4.1.1 HARDWARE INTERFACE

Figure 4-1 is a diagram of a 386 SX CPU/82385SX system, which can be thought of as three distinct interfaces. The first is the 386 SX CPU/82385SX interface (including the Ready Logic). The second is the cache interface, as depicted by the cache control bus in the upper left corner of Figure 4-1. The third is the 82385SX bus interface, which includes both direct connects and signals that control the 74374 address/cycle definition latch and 74646 latching data transceiver. (The 82385SX bus interface is the subject of the next chapter.)

As seen in Figure 4-1, the 386 SX CPU/82385SX interface is a straightforward connection. The only necessary support logic is that required to sum all ready sources.



290222-12

Figure 4-1. 386™ SX CPU/82385SX Interface

4-562

4.1.2 READY GENERATION

Note in Figure 4-1 that the ready logic consists of two gates. The upper three-input AND gate (shown as a negative logic OR) sums all 386 SX local bus ready sources. One such source is the 82385SX READYO# output, which terminates read hits and posted writes. The output of this gate drives the 386 SX READY# input and is monitored by the 82385SX (via READY1#) to track the 386 SX bus state.

When the 82385SX forwards a 386 SX read cycle to the 82385SX bus (cache read miss or non-cacheable read), it does not directly terminate the cycle via READYO#. Instead, the 386 SX and 82385SX bus cycles are concurrently terminated by a system ready source. This is the purpose of the additional two-input OR gate (negative logic AND) in Figure 4-1. When the 82385SX forwards a read to the 82385SX bus, it asserts BRDYEN# which enables the system ready signal (BREADY#) to directly terminate the 386 SX bus cycle.

Figure 4-2A and 4-2B illustrate the behavior of the signals involved in ready generation. Note in cycle 1 of Figure 4-2A that the 82385SX READYO# directly terminates the hit cycle. In cycle 2, READYO# is not activated. Instead the 82385SX BRDYEN# is activated in BT2, BT2P, or BT2I states such that BREADY# can concurrently terminate the 386 SX and 82385SX bus cycles (frame 6). Cycle 3 is a posted write. The write data becomes available in T1P (frame 7), and the address, data, and cycle definition of the write are latched in T2 (frame 8). The 386 SX cycle is terminated by READYO# in frame 8 with no wait states. The 82385SX, however, sees the write cycle through to completion on the 82385SX bus where it is terminated in frame 10 by BREADY#. In this case, the BREADY# signal is not gated through to the 386 SX. Refer to Figures 4-2A and 4-2B for clarification.

4.1.3 NA# AND 386 SX LOCAL BUS PIPELINING

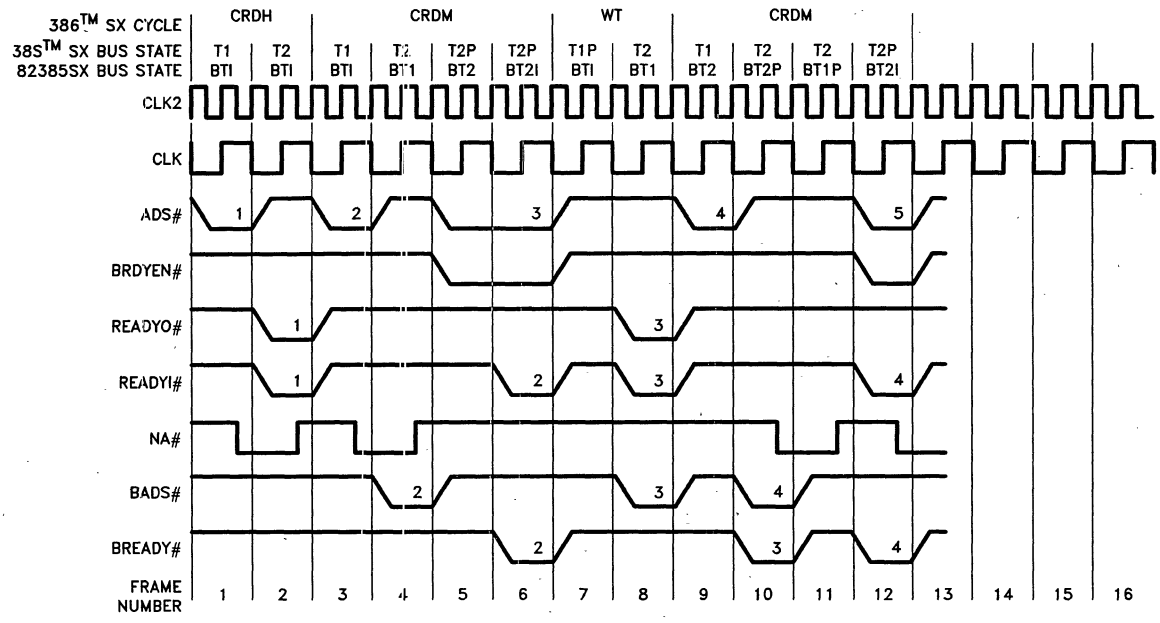
Cycle 1 of Figure 4-2A is a typical cache read hit. The 386 SX address becomes available in T1, and the 82385SX uses this address to determine if the referenced data resides in the cache. The cache look-up is completed and the cycle qualified as a hit or miss in T1. If the data resides in the cache, the cache is directed to drive the 386 SX data bus, and the 82385SX drives its READYO# output so the cycle can be terminated at the end of the first T2 with no wait states.

Although cycle 2 starts out like cycle 1, at the end of T1 (frame 3), it is qualified as a miss and forwarded to the 82385SX bus. The 82385SX bus cycle begins

one state after the 386 SX bus cycle, implying a one wait state overhead associated with cycle 2 due to the look-up. When the 82385SX encounters the miss, it immediately asserts NA#, which puts the 386 SX into pipelined mode. Once in pipelined mode, the 82385SX is able to qualify a 386 SX cycle using the 386 SX pipelined address and control signals. The result is that the cache look-up state is hidden in all but the first of a contiguous sequence of read misses. This is shown in the first two cycles, both read misses, of Figure 4-2B. The CPU sees the look-up state in the first cycle, but not in the second. In fact, the second miss requires a total of only two states, as not only does 386 SX pipelining hide the look-up state, but system pipelining hides one of the main memory wait states. (System level pipelining via BNA# is discussed in the next chapter.) Several characteristics of the 82385SX's pipelining of the 386 SX are as follows:

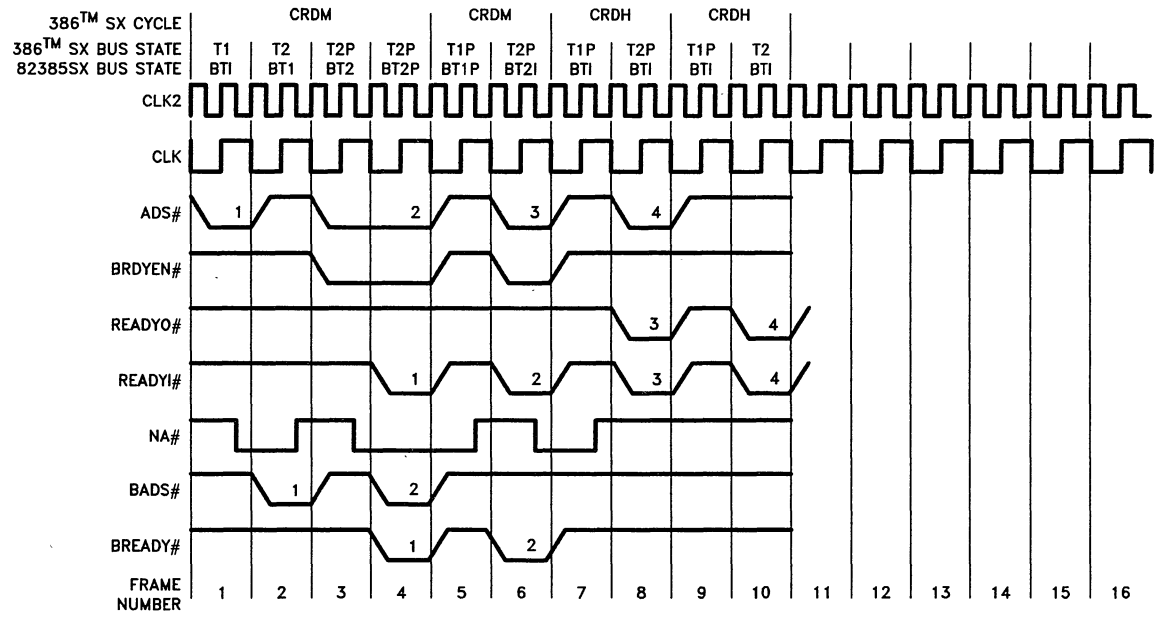
- The above discussion applies to all system reads, not just cache read misses.
- The 82385SX provides the fastest possible switch to pipelining, T1-T2-T2P. The exception to this is when a system read follows a posted write. In this case, the sequence is T1-T2-T2-T2P. (Refer to cycle 4 of Figure 4-2A.) The number of T2 states is dependent on the number of main memory wait states.
- Refer to the read hit in Figure 4-2A (cycle 1), and note that NA# is actually asserted before the end of T1, before the hit/miss decision is made. This is of no consequence since even though NA# is sampled active in T2, the activation of READYO# in the same T2 renders NA# a "don't care". NA# is asserted in this manner to meet 386 SX timing requirements and to insure the fastest possible switch to pipelined mode.
- All read hits and the majority of writes can be serviced by the 82385SX with zero wait states in non-pipelined mode, and the 82385SX accordingly attempts to run all such cycles in non-pipelined mode. An exception is seen in the hit cycles (cycles 3 and 4) of Figure 4-2B. The 82385SX does not know soon enough that cycle 3 is a hit, and thus sustains the pipeline. The result is that three sequential hits are required before the 386 SX is totally out of pipelined mode. (The three hits look like T1P-T2P, T1P-T2, T1-T2.) Note that this does not occur if the number of main memory wait states is equal to or greater than two.

As far as the design is concerned, NA# is generally tied directly to the 386 SX NA# input. However, other local NA# sources may be logically "AND"ed with the 82385SX NA# output if desired. It is essential, however, that no device other than the 82385SX drive the 386 SX NA# input unless that device re-



290222-13

Figure 4-2A. READY#, BRDYEN#, and NA# (N = 1)
4-564



290222-14

Figure 4-2B. READY#, BRDYEN#, and NA# (N = 1)

4-565

sides on the 386 SX local bus in space decoded via LBA#. If desired, the 82385SX NA# output can be ignored and the 386 SX NA# input tied high. The 386 SX NA# input should never be tied low, which would always keep it active.

4.1.4 LBA# AND NCA# GENERATION

The 82385SX inputs signals LBA# and NCA# are generated by decoding the 386 SX address (A1-A23) and cycle definition (W/R#, D/C#, M/IO#) lines. The 82385SX samples them at the end of the first state in which they become available, which is either T1 or the first T2P cycle. The decode configuration and timings are illustrated respectively in Figures 4-3A and 4-3B.

4.2 Cache Interface

The following is a description of the external data cache and 82385SX cache interface.

4.2.1 CACHE CONFIGURATIONS

The 82385SX controls the cache memory via the control signals shown in Figure 4-1. These signals drive one of four possible cache configurations, as depicted in Figures 4-4A through 4-4D. Figure 4-4A shows a direct mapped cache organized as 8K words. The likely design choice is two 8K x 8 SRAMs. Figure 4-4B depicts the same cache memory but with a data transceiver between the cache and 386 SX data bus. In this configuration, CT/R# controls the transceiver direction, COEA# drives the transceiver output enable (COEB# could also be used), and DEFOE# is strapped high. A data buffer is required if the chosen SRAM does not have a separate output enable. Additionally, buffers may be used to ease SRAM timing requirements or in a system with a heavily loaded data bus. (Guidelines for SRAM selection are included in Chapter 6.)

Figure 4-4C depicts a two-way set associative cache organized as two banks (A and B) of 4K words each. The likely design choice is eight 4K x 4 SRAMs. Finally, Figure 4-4D depicts the two-way organization with data buffers between the cache memory and data bus.

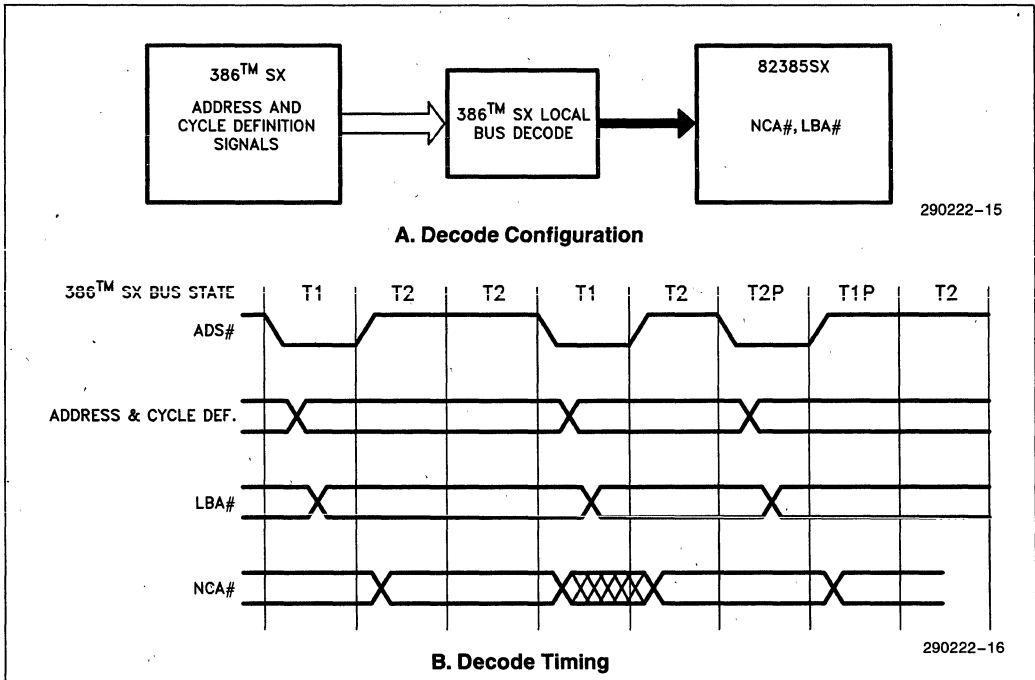


Figure 4-3. NCA#, LBA# Generation

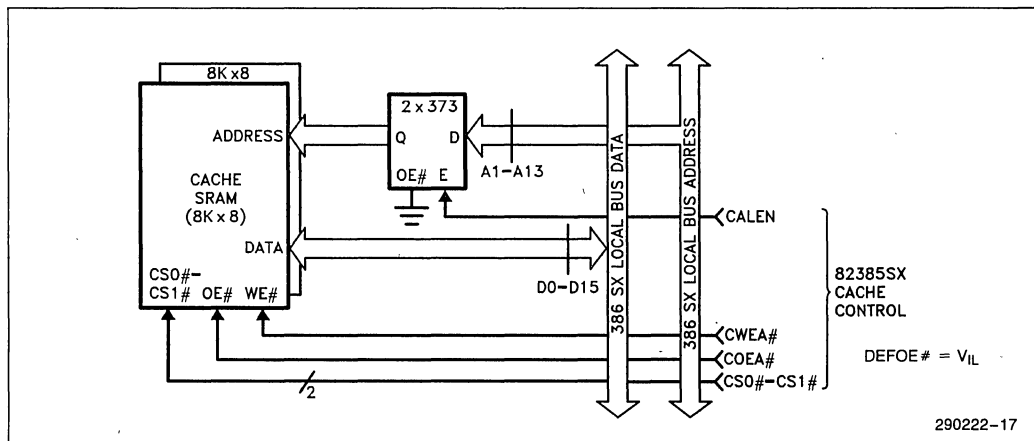


Figure 4-4A. Direct Mapped Cache without Data Buffers

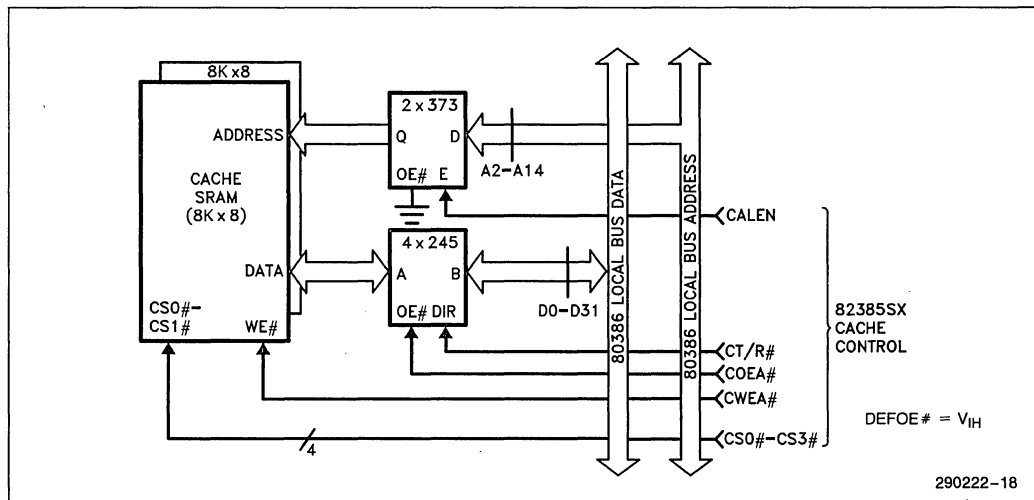


Figure 4-4B. Direct Mapped Cache with Data Buffers

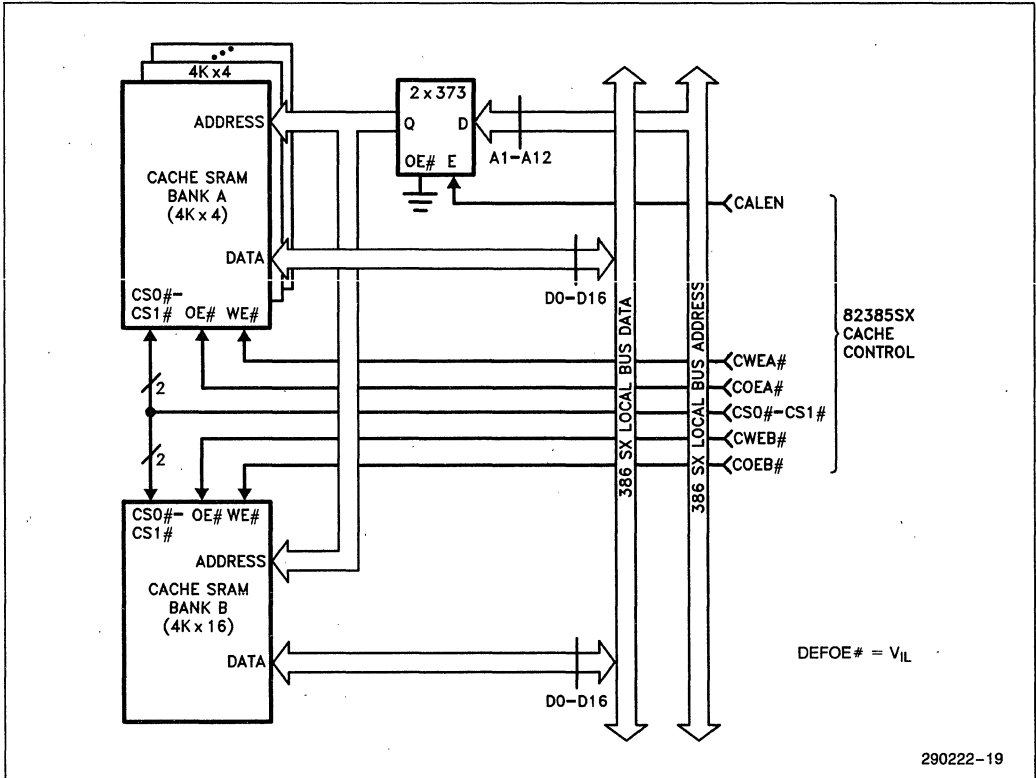


Figure 4-4C. Two-Way Set Associative Cache without Data Buffers

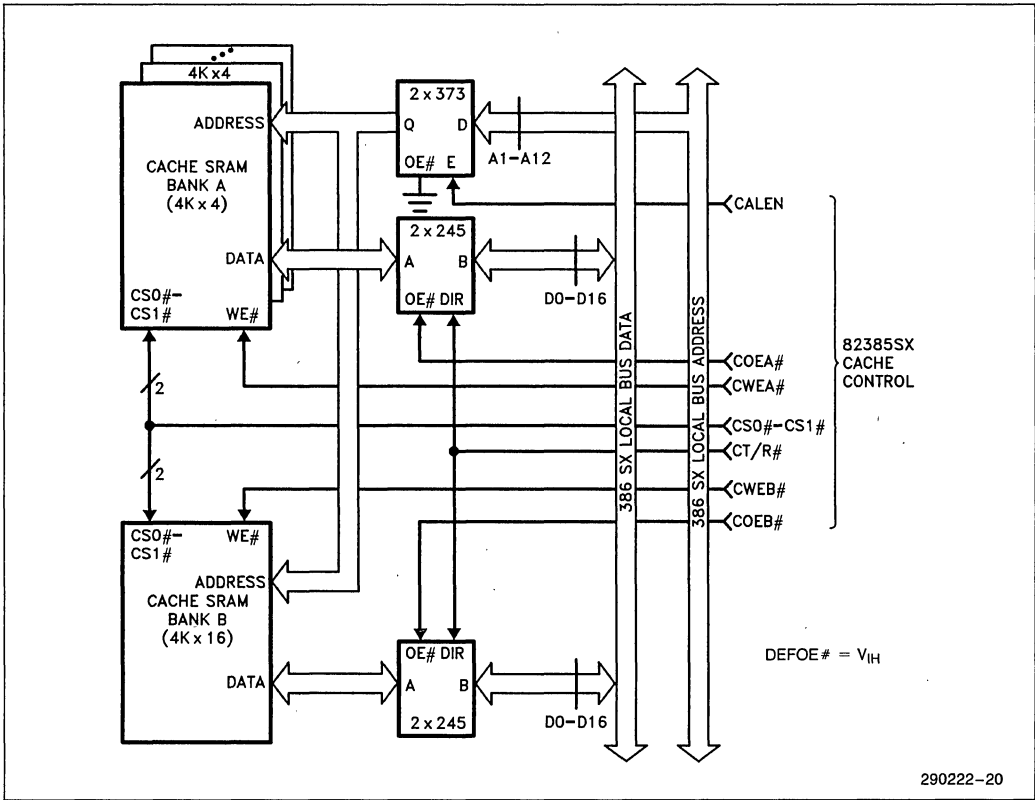
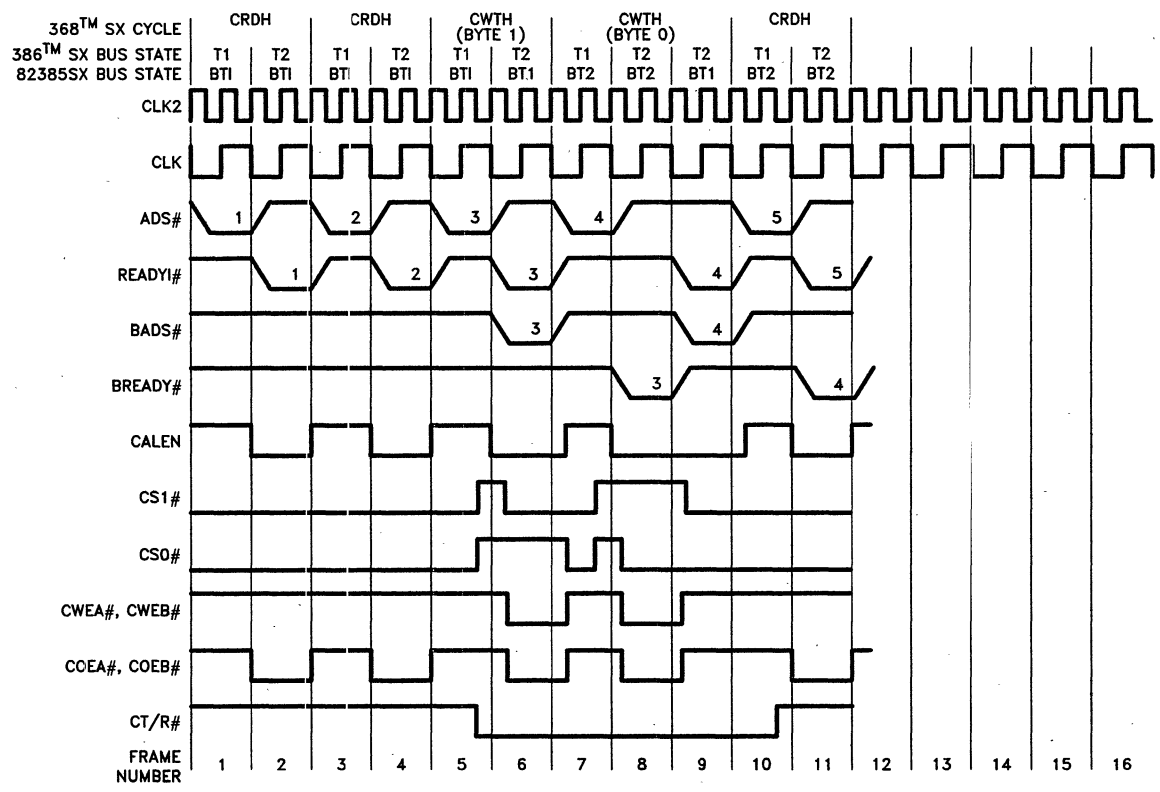


Figure 4-4D. Two-Way Set Associative Cache with Data Buffers

4.2.2 CACHE CONTROL ... DIRECT MAPPED

Figure 4-5A illustrates the timing of cache read and write hits, while Figure 4-5B illustrates cache updates. In a read hit, the cache output enables are driven from the beginning of T2 (cycle 1 of Figure 4-5A). If at the end of T1 the cycle is qualified as a cacheable read, the output enables are asserted on the assumption that the cycle will be a hit. (Driving the output enables before the actual hit/miss decision is made eases SRAM timing requirements.)

Cycle 1 of Figure 4-5B illustrates what happens when the assumption of a hit turns out to be wrong. Note that the output enables are asserted at the beginning of T2, but then disabled at the end of T2. Once the output enables are inactive, the 82385SX turns the transceiver around (via CT/R#) and drives the write enables to begin the cache update cycle. Note in Figure 4-5B that once the 386 SX is in pipelined mode, the output enables need not be driven prior to a hit/miss decision, since the decision is made earlier via the pipelined address information.



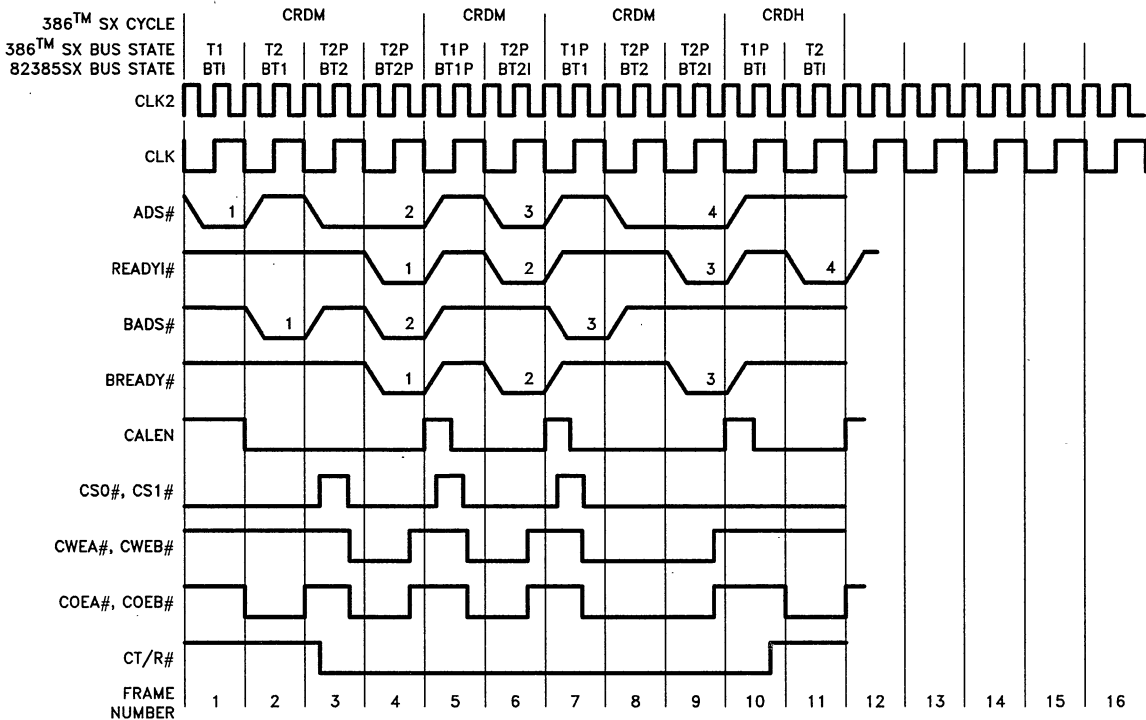
NOTES:
 CRDH = Cache Read Hit
 CWTH = Cache Write Hit

N = Number of Non-Pipelined, main memory wait states. Must be greater than zero.

290222-21

Figure 4-5A. Cache Read and Write Cycles—Direct Mapped (N = 1)

4-570



290222-22

N = Number of Non-Pipelined, main memory wait states. Must be greater than zero.

NOTE:
CRDM = Cache Read Miss

Figure 4-5B. Cache Update Cycles—Direct Mapped (N = 1)

One consequence of driving the output enables low in a miss before the hit/miss decision is made is that since the cache starts driving the 386 SX data bus, the 82385SX cannot enable the 74646 transceiver (Figure 4-1) until after the cache outputs are disabled. (The timing of the 74646 control signals is described in the next chapter.) The result is that the 74646 cannot be enabled soon enough to support $N=0$ main memory ("N" was defined in Section 4.0 as the number of non-pipelined main memory wait states). This means that memory which can run with zero wait states in a non-pipelined cycle should not be mapped into cacheable memory. This should not present a problem, however, as a main memory system built with $N=0$ memory has no need of a cache. (The main memory is as fast as the cache.) Zero wait state memory can be supported if it is decoded as non-cacheable. The 82385SX knows that a cycle is non-cacheable in time not to drive the cache output enables, and can thus enable the 74646 sooner.

In a write hit, the 82385SX only updates the cache bytes that are meant to be updated as directed by the 386 SX byte enables. This prevents corrupting cache data in partial doubleword writes. Note in Figure 4-5A that the appropriate bytes are selected via the cache byte select lines CS0# and CS1#. In a read hit, both select lines are driven as the 386 SX will simply ignore data it does not need. Also, in a cache update (read miss), both selects are active in order to update the cache with a complete line (word).

4.2.3 CACHE CONTROL ... TWO-WAY SET ASSOCIATIVE

Figures 4-6A and 4-6B illustrate the timing of cache read hits, write hits, and updates for a two-way set associative cache. (Note that the cycle sequences are the same as those in Figure 4-5A and 4-5B.) In a cache read hit, only one bank on the other is enabled to drive the 386 SX data bus, so unlike the control of a direct mapped cache, the appropriate cache output enable cannot be driven until the outcome of the hit/miss decision is known. (This implies stricter SRAM timing requirements for a two-way set associative cache.) In write hits and read misses, only one bank or the other is updated.

4.3 387 SX Interface

The 387 SX Math Coprocessor interfaces to the 386 SX just as it would in a system without an 82385SX. The 387 SX READY0# output is logically "AND"ed along with all other 386 SX local bus ready sources (Figure 4-1), and the output is fed to the 387 SX READY#, 82385SX READY1#, and 386 SX READY# inputs.

The 386 SX uniquely addresses the 387 SX by driving M/IO# low and A23 high. The 82385SX decodes this internally and treats 387 SX accesses in the same way it treats 386 SX cycles in which LBA# is asserted, it ignores them.

5.0 82385SX LOCAL BUS AND SYSTEM INTERFACE

The 82385SX system interface is the 82385SX Local Bus, which presents a "386 SX-like" front end to the system. The system ties to it just as it would to a 386 SX. Although this 386 SX-like front end is functionally equivalent to a 386 SX, there are timing differences which can easily be accounted for in a system design.

The following is a description of the 82385SX system interface. After presenting the 82385SX bus state machine, the 82385SX bus signals are described, as are techniques for accommodating any differences between the 82385SX bus and 386 SX bus. Following this is a discussion of the 82385SX's condition upon reset.

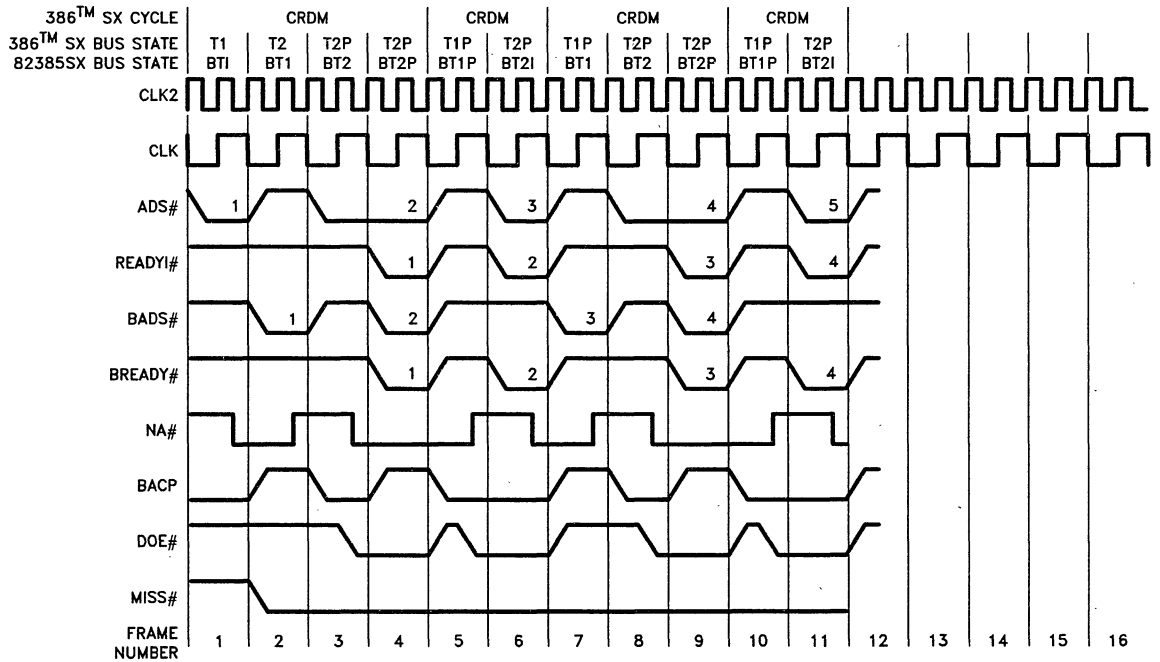
5.1 The 82385SX Bus State Machine

5.1.1 MASTER MODE

Figure 5-1A illustrates the 82385SX bus state machine when the 82385SX is programmed in master mode. Note that it is almost identical to the 386 SX bus state machine, only the bus states are 82385SX bus states (BT1P, BTH, etc.) and the state transitions are conditioned by 82385SX bus inputs (BNA# BHOLD, etc.). Whereas a "pending request" to the 386 SX state machine indicates that the 386 SX execution or prefetch unit needs bus access, a pending request to the 82385SX state machine indicates that a 386 SX bus cycle needs to be forwarded to the system (read miss, non-cacheable read, write, etc.). The only difference between the state machines is that the 82385SX does not implement a direct BT1P-BT2P transition. If BNA# is asserted in BT1P, the resulting state sequence is BT1P-BT2I-BT2P. The 82385SX's ability to sustain a pipeline is not affected by the lack of this transition.

5.1.2 SLAVE MODE

The 82385SX's slave mode state machine (Figure 5-1B) is similar to the master mode machine except that now transitions are conditioned by BHLDA rather than BHOLD. (Recall that in slave mode, the roles of BHOLD and BHLDA are reversed from their master mode roles.) Figure 5-2 clarifies slave mode state machine operation. Upon reset, a slave mode

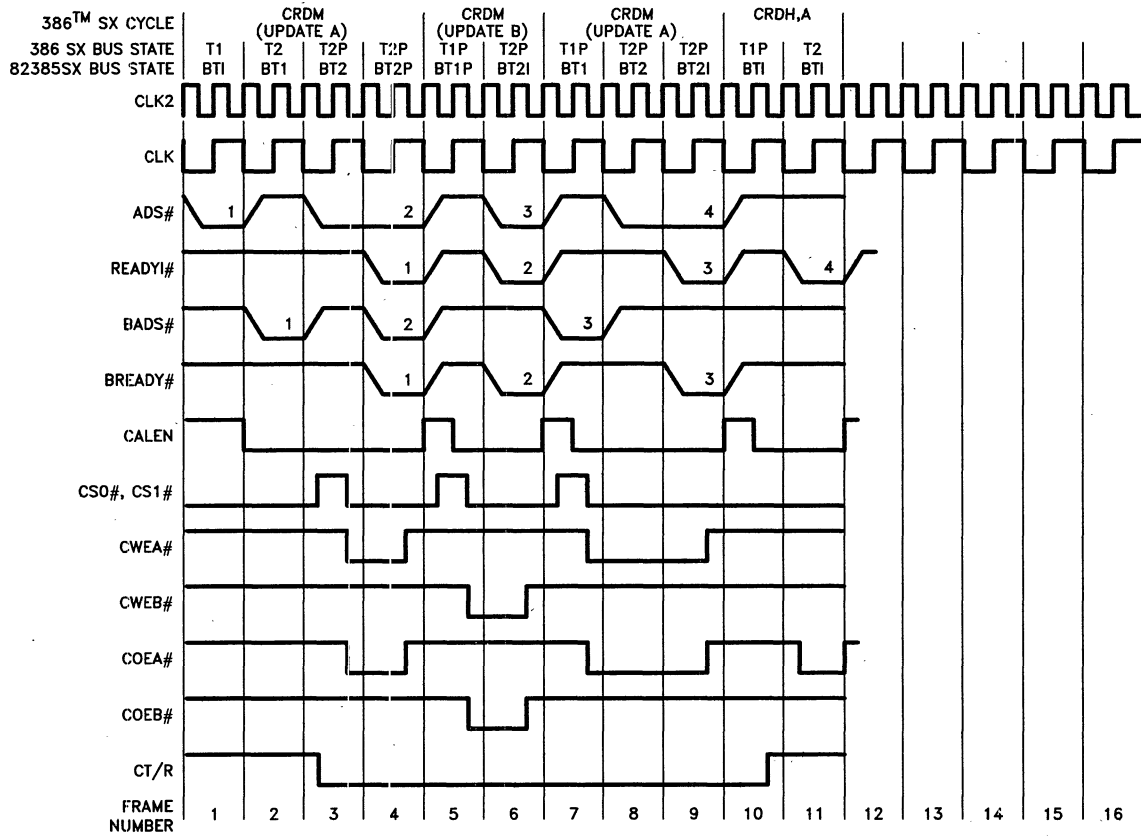


N = Number of Non-Pipelined, main memory wait states. Must be greater than zero.

290222-23

Figure 4-6A. Cache Read and Write Cycles—Two Way Associative (N = 1)

4-573



N = Number of Non-Pipelined, main memory wait states. Must be greater than zero.

290222-24

Figure 4-6B. Cache Update Cycles—Two Way Set Associative (N = 1)

4-5/4

82385SX enters the BTH state. When the 386 SX of the slave 82385SX subsystem has a cycle that needs to be forwarded to the system, the 82385SX moves to BTI and issues a hold request via BHOLD. It is important to note that a slave mode 82385SX does not drive the bus in a BTI state. When the master or bus arbiter returns BHLDA, the slave 82385SX enters BT1 and runs the cycle. When the cycle is completed, and if no additional requests are pending, the 82385SX moves back to BTH and disables BHOLD.

If, while a slave 82385SX is running a cycle, the master or arbiter drops BHLDA (Figure 5-2B), the 82385SX will complete the current cycle, move to BTH and remove the BHOLD request. If the 82385SX still had cycles to run when it was kicked off the bus, it will immediately assert a new BHOLD and move to BTI to await bus acknowledgement. Note, however, that it will only move to BTI if BHLDA is negated, insuring that the handshake sequence is completed.

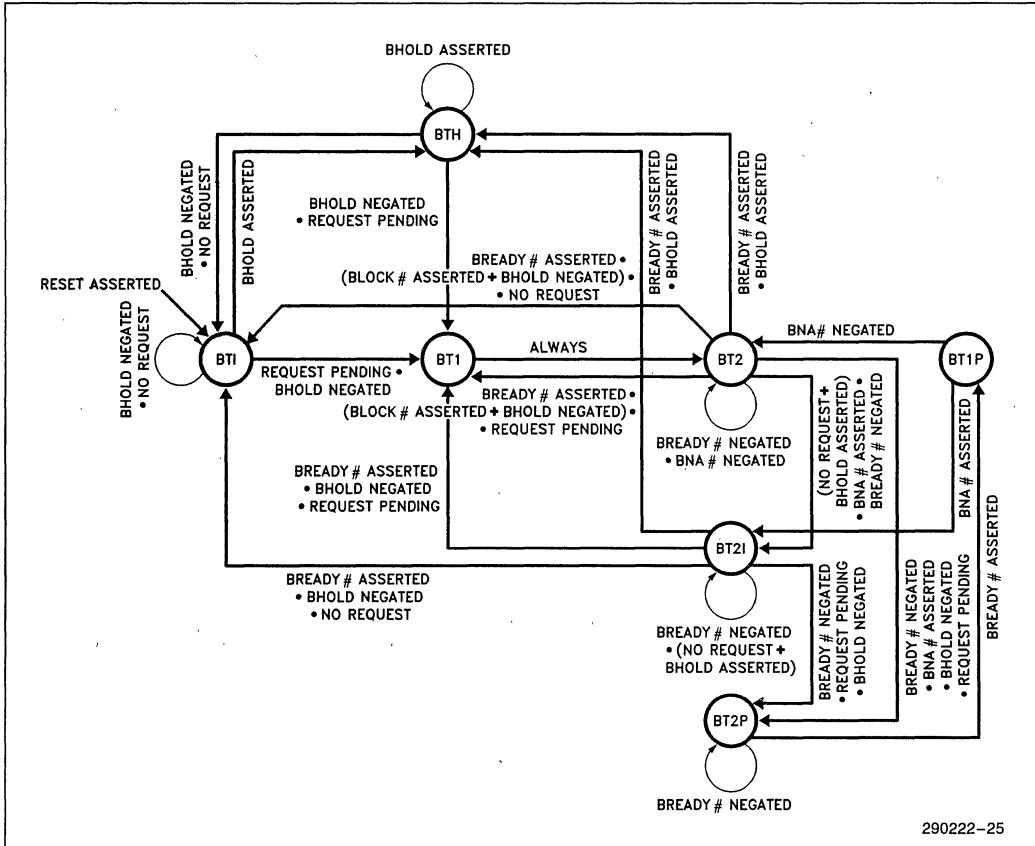
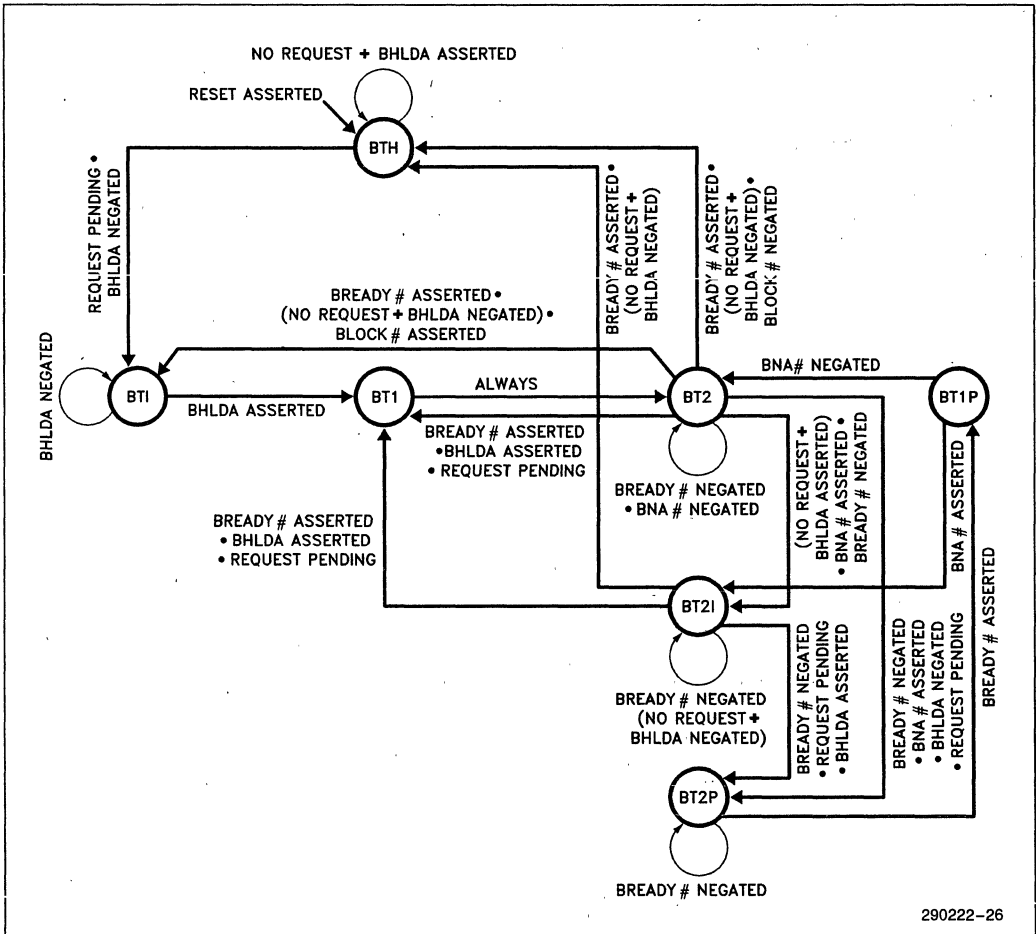


Figure 5-1A. 82385SX Local Bus State Machine—Master Mode



290222-26

Figure 5-1B. 82385SX Local Bus State Machine—Slave Mode

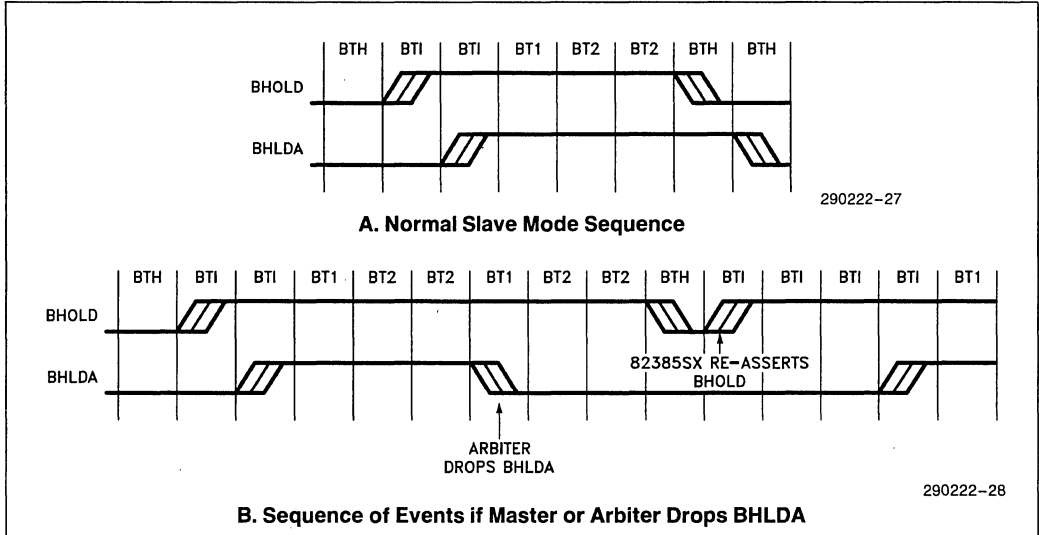


Figure 5-2. B HOLD/BHLDA—Slave Mode

There are several cases in which a slave 82385SX will not immediately release the bus if BHLDA is dropped. For example, if BHLDA is dropped during a BT2P state, the 82385SX has already committed to the next system bus pipelined cycle and will execute it before releasing the bus. Also, the 82385SX will complete a sequence of locked cycles before releasing the bus. This should not present any problems, as a properly designed arbiter will not assume that the 82385SX has released the bus until it sees B HOLD become inactive.

5.2 The 82385SX Local Bus

The 82385SX bus can be broken up into two groups of signals: those which have direct 386 SX counterparts, and additional status and control signals provided by the 82385SX. The operation and interaction of all 82385SX bus signals are depicted in Figures 5-3A through 5-3L for a wide variety of cycle sequences. These diagrams serve as a reference for the 82385SX bus discussion and provide insight into the dual bus operation of the 82385SX.

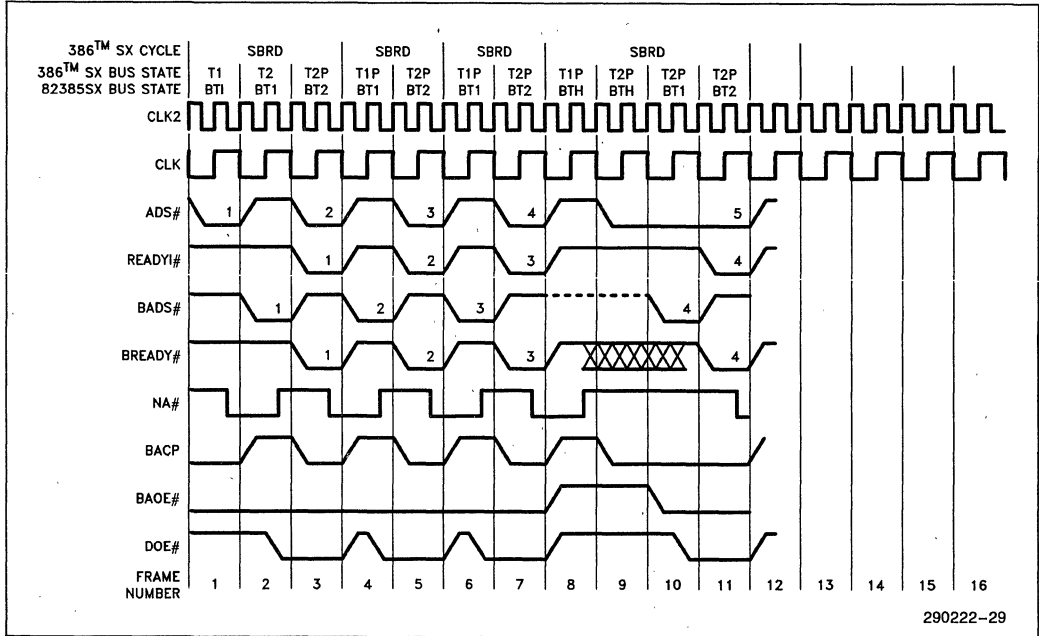


Figure 5-3A. Consecutive SBRD Cycles—(N = 0)

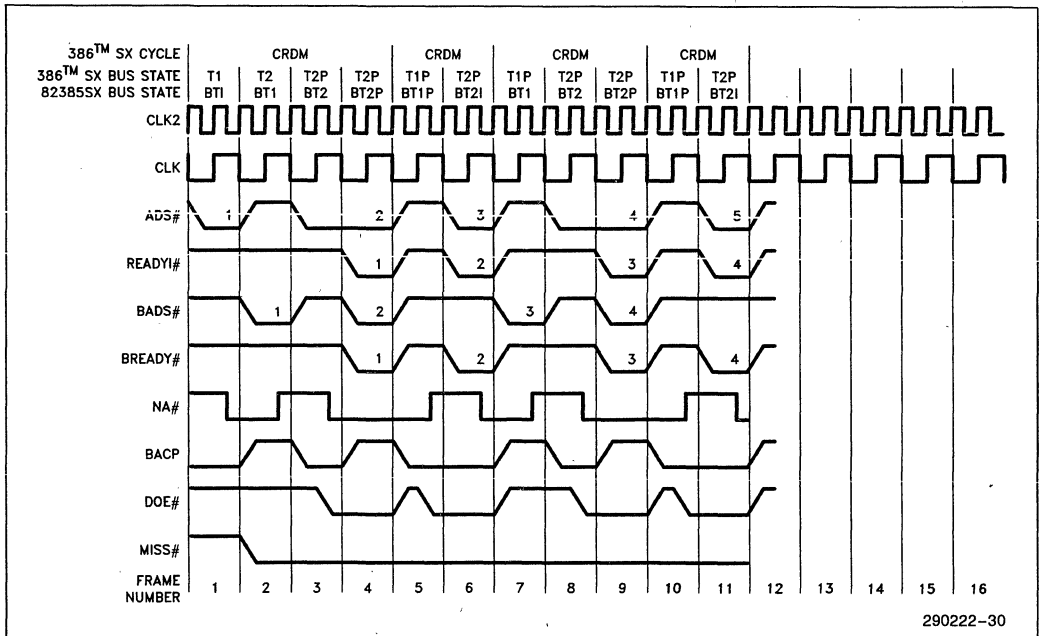


Figure 5-3B. Consecutive CRDM Cycles—(N = 1)

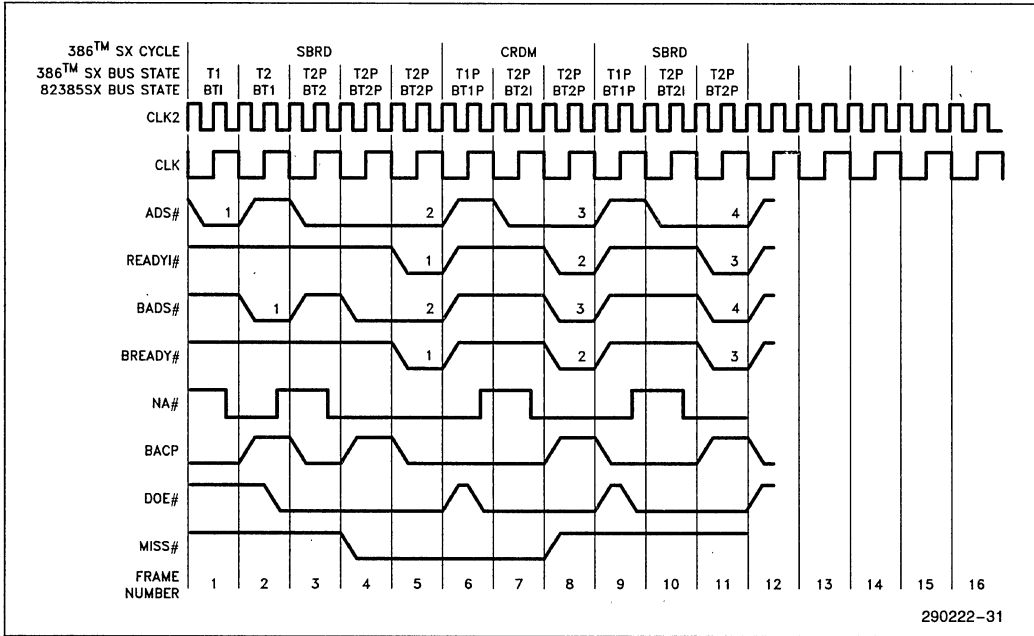


Figure 5-3C. SBRD, CRDM, SBRD—(N = 2)

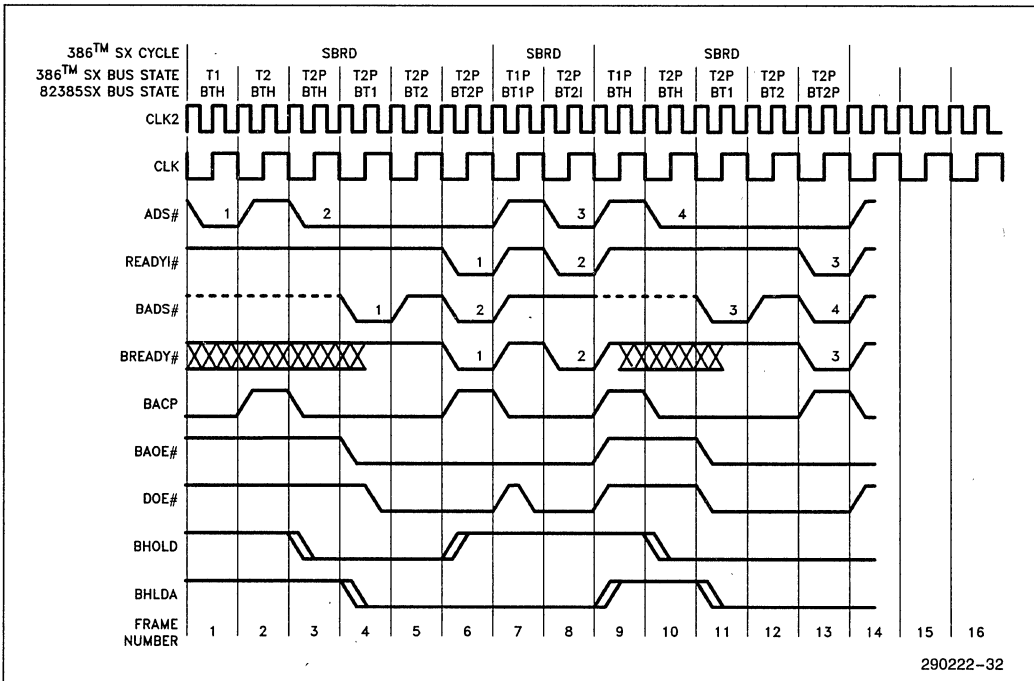


Figure 5-3D. SBRD Cycles Interleaved with BTH States—(N = 1)

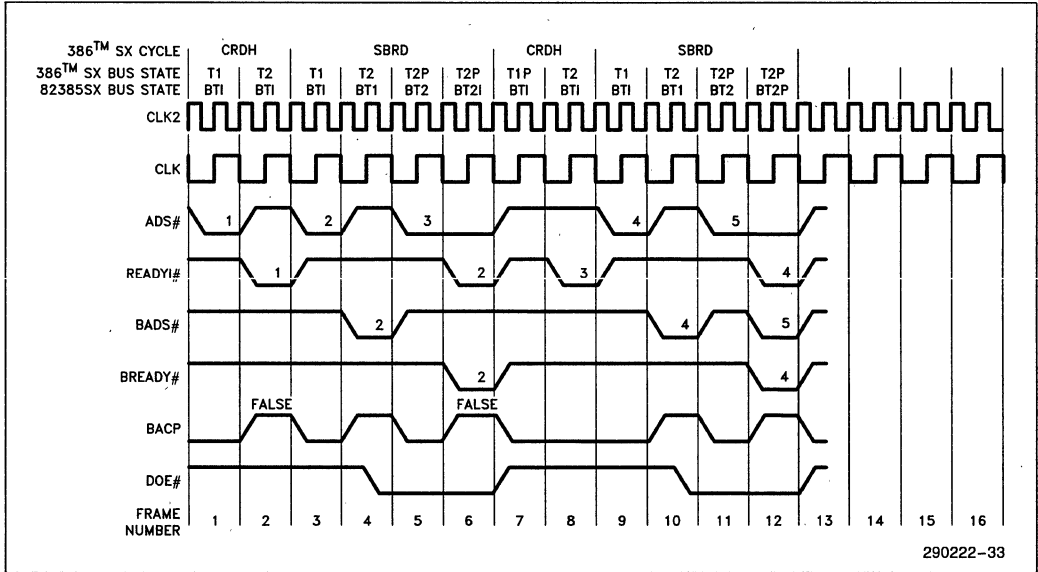


Figure 5-3E. Interleaved SBRD/CDRH Cycles—(N = 1)

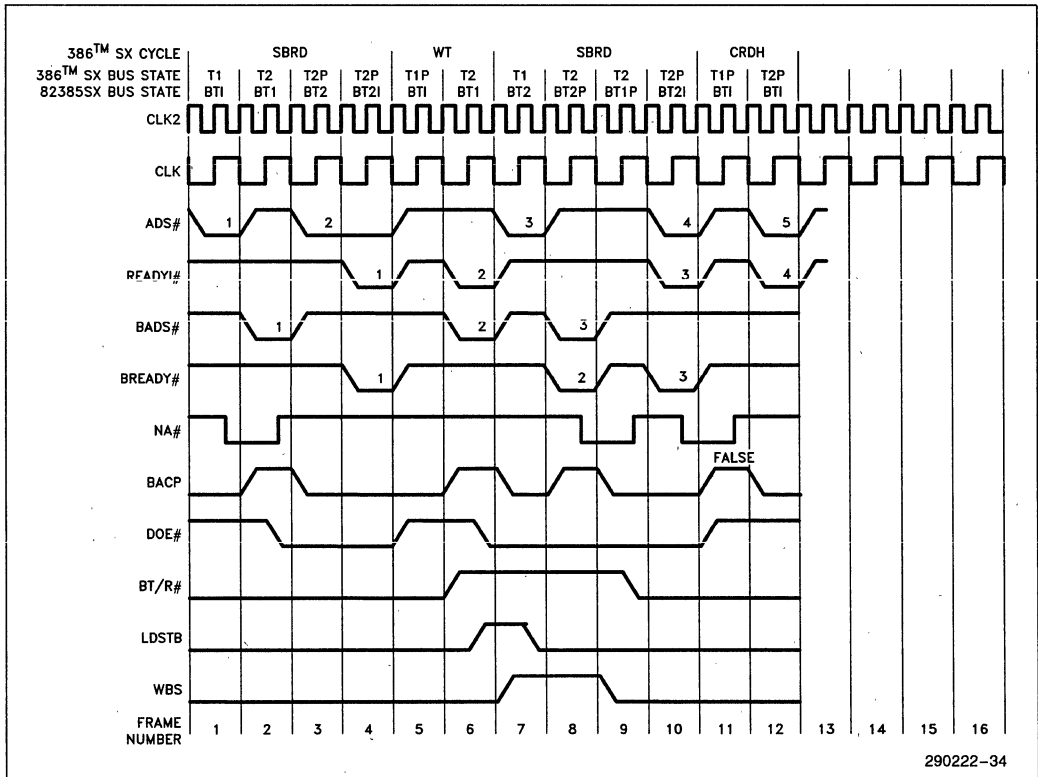


Figure 5-3F. SBRD, WT, SBRD, CRDH—(N = 1)

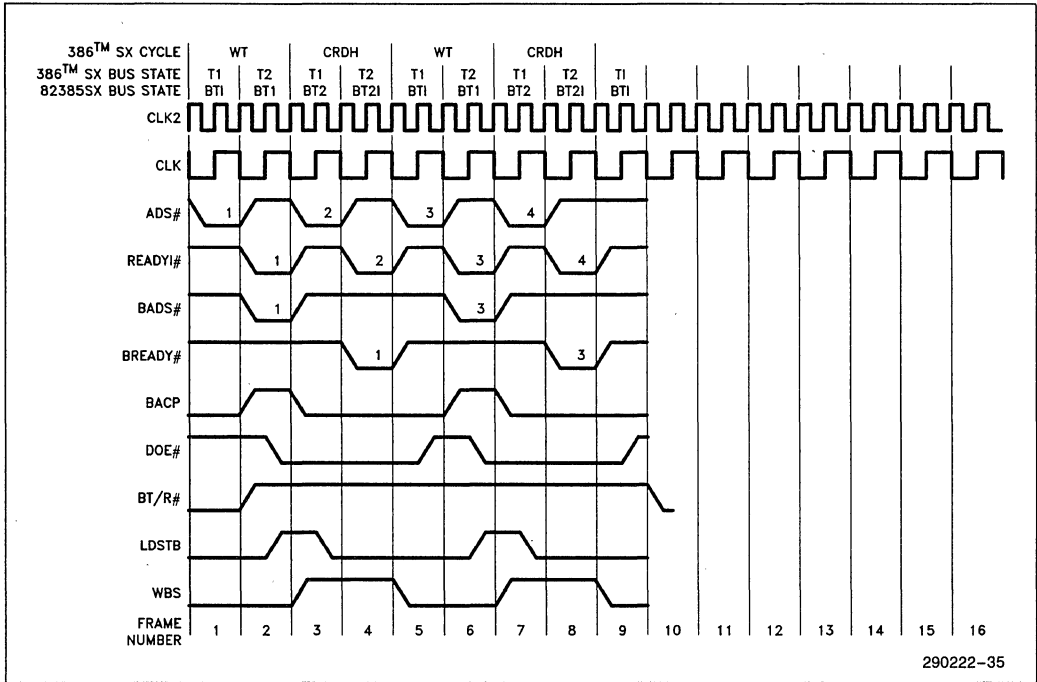


Figure 5-3G. Interleaved WT/CRDH Cycles—(N = 1)

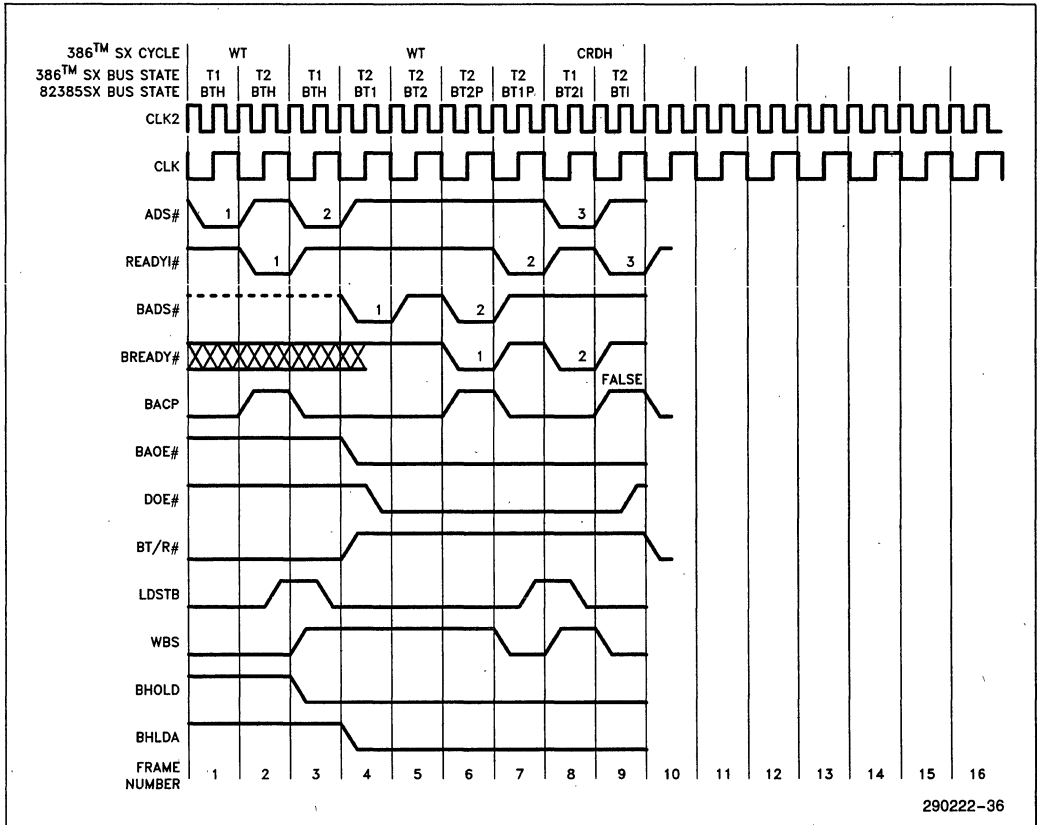
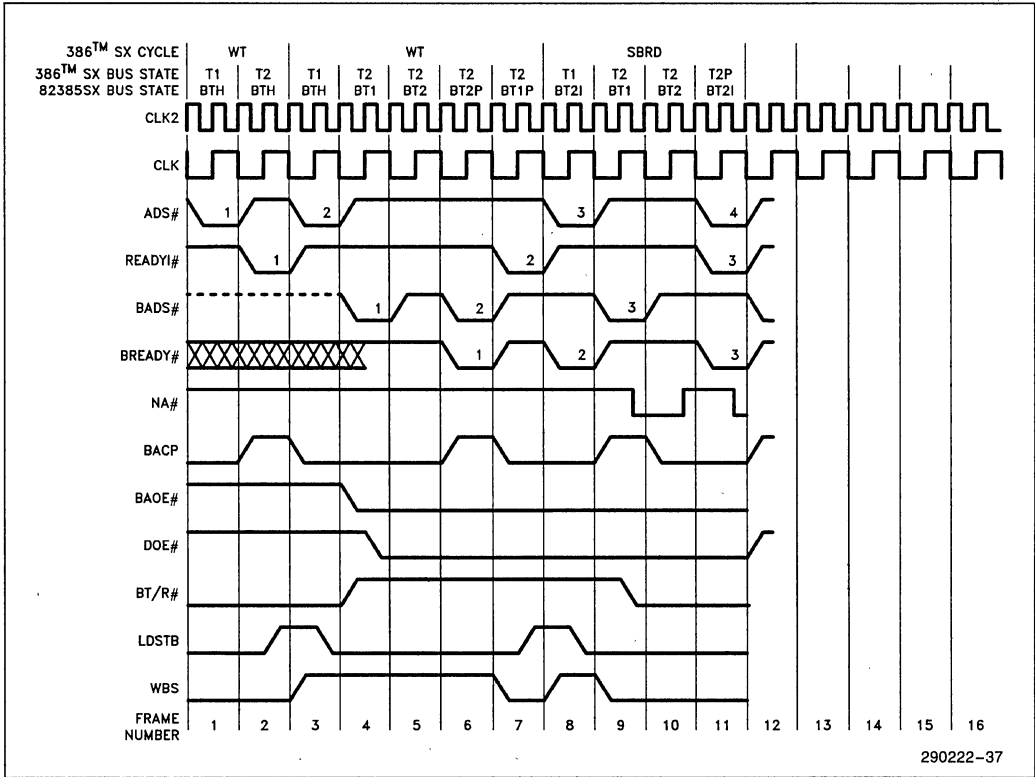


Figure 5-3H. WT, WT, CRDH—(N = 1)

290222-36



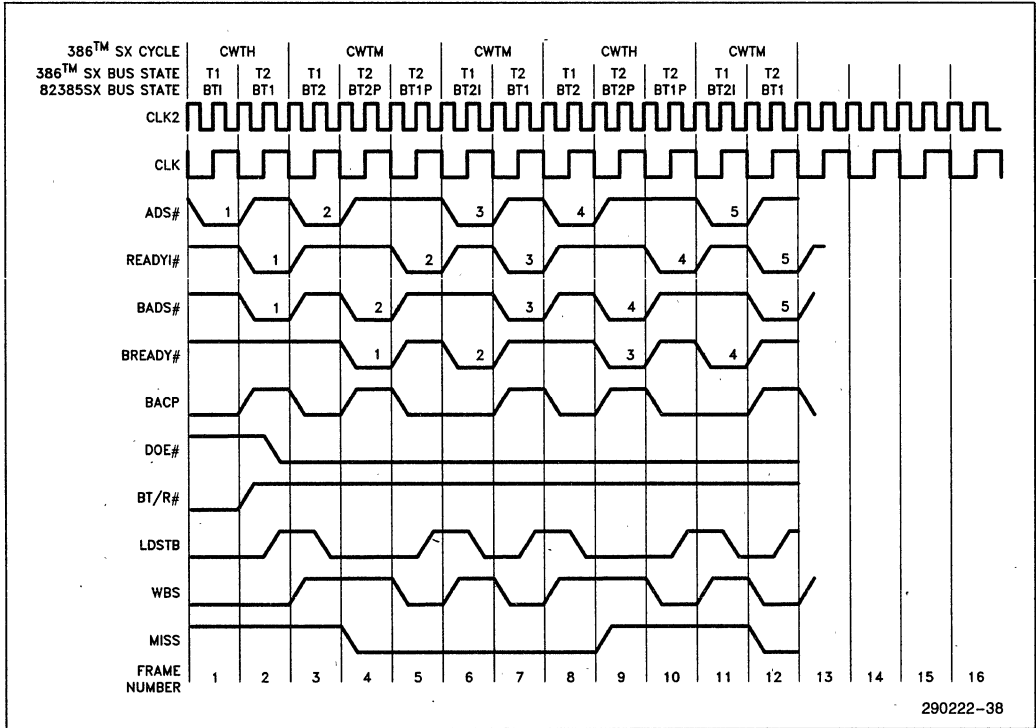


Figure 5-3J. Consecutive Write Cycles—(N = 1)

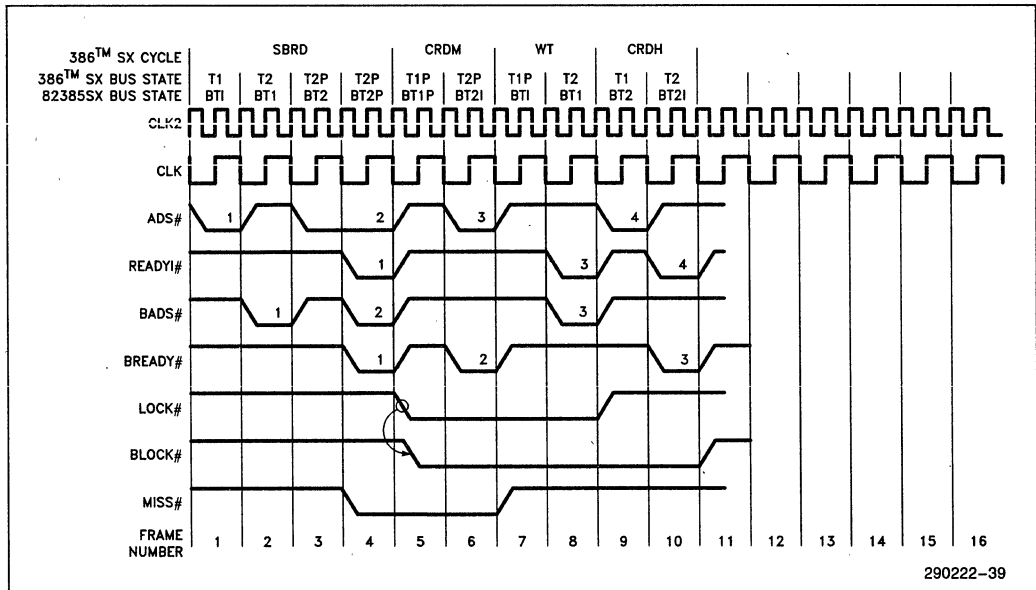


Figure 5-3K. LOCK #/BLOCK # in Non-Cacheable or Miss Cycles—(N = 1)

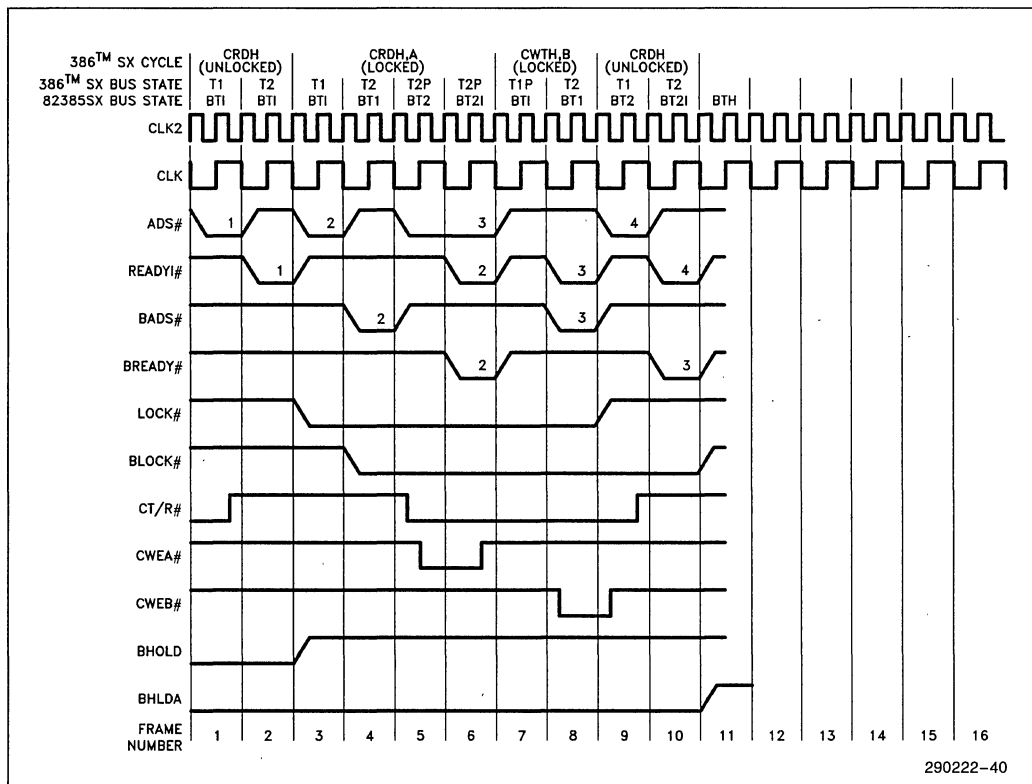


Figure 5-3L. LOCK # /BLOCK # in Cache Read Hit Cycle—(N = 1)

5.2.1 82385SX BUS COUNTERPARTS TO 386™ SX SIGNALS

The following sections discuss the signals presented on the 82385SX local bus which are functional equivalents to the signals present at the 386 SX local bus.

5.2.1.1 Address Bus (BA1-BA23) and Cycle Definition Signals (BM/IO#, BD/C#, BW/R#)

These signals are not driven directly by the 82385SX, but rather are the outputs of the 74374 address/cycle definition latch. (Refer to Figure 4-1 for the hardware interface.) This latch is controlled by the 82385SX BACP and BAOE# outputs. The behavior and timing of these outputs and the latch they control (typically F or AS series TTL) ensure that BA1-BA23, BM/IO#, BW/R#, and BD/C# are compatible in timing and function to their 386 SX counterparts.

The behavior of BACP can be seen in Figure 5-3B, where the rising edge of BACP latches and forwards the 386 SX address and cycle definition signals in a BT1 or first BT2P state. However, the 82385SX need not be the current bus master to latch the 386 SX address, as evidenced by cycle 4 of Figure 5-3A. In this case, the address is latched in frame 8, but not forwarded to the system (via BAOE#) until frame 10. (The latch and output enable functions of the 74374 are independent and invisible to one another.)

Note that in frames 2 and 6 the BACP pulses are marked "False". The reason is that BACP is issued and the address latched before the hit/miss determination is made. This ensures that should the cycle be a miss, the 82385SX bus can move directly into BT1 without delay. In the case of a hit, the latched address is simply never qualified by the assertion of BADS#. The 82385SX bus stays in BT1 if there is no access pending (new cycle is a hit) and no bus activity. It will move to and stay in BT2I if the system has requested a pipelined cycle and the 82385SX does not have a pending bus access (new cycle is a hit).

5.2.1.2 Data Bus (BD0–BD15)

The 82385SX data bus is the system side of the 74646 latching transceiver. (See Figure 4-1.) This device is controlled by the 82385SX outputs LDSTB, DOE#, and BT/R#. LDSTB latches data in write cycles, DOE# enables the transceiver outputs, and BT/R# controls the transceiver direction. The interaction of these signals and the transceiver is such that BD0–BD15 behave just like their 386 SX counterparts. The transceiver is configured such that data flow in write cycles (A to B) is latched, and data flow in read cycles (B to A) is flow-through.

Although BD0–BD15 function just like their 386 SX counterparts, there is a timing difference that must be accommodated for in a system design. As mentioned above, the transceiver is transparent during read cycles, so the transceiver propagation delay must be added to the 386 SX data setup. In addition, the cache SRAM setup must be accommodated for in cache read miss cycles.

For non-cacheable reads the data setup is given by:

$$\text{Min BD0–BD15 ReadDataSetup} = \frac{386\text{SX Min Data Setup}}{\text{Data Setup}} + \frac{74646\text{B-to-A Max Propagation Delay}}{\text{Max Propagation Delay}}$$

The required BD0–BD15 setup in a cache read miss is given by:

$$\begin{aligned} \text{Min BD0–BD15 ReadDataSetup} &= \frac{74646\text{B-to-A Max Propagation Delay}}{\text{Max Propagation Delay}} + \frac{\text{CacheSRAM Min Write Setup}}{\text{Write Setup}} \\ &+ \frac{\text{One CLK2 Period} - 82385\text{SX CWEA\# or CWEB\# Min Delay}}{\text{Period}} \end{aligned}$$

If a data buffer is located between the 386 SX data bus and the cache SRAMs, then its maximum propagation delay must be added to the above formula as well. A design analysis should be completed for every new design to determine actual margins.

A design can accommodate the increased data setup by choosing appropriately fast main memory DRAMs and data buffers. Alternatively, a designer may deal with the longer setup by inserting an extra wait state into cache read miss cycles. If an additional state is to be inserted, the system bus controller should sample the 82385SX MISS# output to distinguish read misses from cycles that do not require the longer setup. Tips on using the 82385SX MISS# signal are presented later in this chapter.

The behavior of LDSTB, DOE#, and BT/R# can be understood via Figures 5-3A through 5-3L. Note that in cycle 1 of Figure 5-3A (A non-cacheable system read), DOE# is activated midway through BT1, but in cycle 1 of Figure 5-3B (a cache read miss), DOE# is not activated until midway through BT2. The rea-

son is that in a cacheable read cycle, the cache SRAMs are enabled to drive the 386 SX data bus before the outcome of the hit/miss decision (in anticipation of a hit.) In cycle 1 of Figure 5-3B, the assertion of DOE# must be delayed until after the 82385SX has disabled the cache output buffers. The result is that N=0 main memory should not be mapped into the cache.

5.2.1.3 Byte Enables (BBHE#, BBLE#)

These outputs are driven directly by the 82385SX, and are completely compatible in timing and function with their 386 SX counterparts. When a 386 SX cycle is forwarded to the 82385SX bus, the 386 SX byte enables are duplicated on BBHE# and BBLE#. The one exception is a cache read miss, during which BBHE# and BBLE# are both active regardless of the status of the 386 SX byte enables. This ensures that the cache is updated with a valid 16-bit entry.

5.2.1.4 Address Status (BADs#)

BADs# is identical in function and timing to its 386 SX counterpart. It is asserted in BT1 and BT2P states, and indicates that valid address and cycle definition (BA1–BA23, BBHE#, BBLE#, BM/IO#, BW/R#, BD/C#) information is available on the 82385SX bus.

5.2.1.5 Ready (BREADY#)

The 82385SX BREADY# input terminates 82385SX bus cycles just as the 386 SX READY# input terminates 386 SX bus cycles. The behavior of BREADY# is the same as that of READY#, but note in the A.C timing specifications that a cache read miss requires a longer BREADY# setup than do other cycles. This must be accommodated for in ready logic design.

5.2.1.6 Next Address (BNA#)

BNA# is identical in function and timing to its 386 SX counterpart. Note that in Figures 5-3A through 5-3L, BNA# is assumed asserted in every BT1P or first BT2 state. Along with the 82385SX's pipelining of the 386 SX, this ensures that the timing diagrams accurately reflect the full pipelined nature of the dual bus structure.

5.2.1.7 Bus Lock (BLOCK#)

The 386 SX flags a locked sequence of cycles by asserting LOCK#. During a locked sequence, the 386 SX does not acknowledge hold requests, so the

sequence executes without interruption by another master. The 82385SX forces all locked 386 SX cycles to run on the 82385SX bus (unless LBA# is active), regardless of whether or not the referenced location resides in the cache. In addition, a locked sequence of 386 SX cycles is run as a locked sequence on the 82385SX bus; BLOCK# is asserted and the 82385SX does not allow the sequence to be interrupted. Locked writes (hit or miss) and locked read misses affect the cache and cache directory just as their unlocked counterparts do. A locked read hit, however, is handled differently. The read is necessarily forced to run on the 82385SX local bus, and as the data returns from main memory, it is "re-copied" into the cache. (See Figure 5-3L.) The directory is not changed as it already indicates that this location exists in the cache. This activity is invisible to the system and ensures that semaphores are properly handled.

BLOCK# is asserted during locked 82385SX bus cycles just as LOCK# is asserted during locked 386 SX cycles. The BLOCK# maximum valid delay, however, differs from that of LOCK#, and this must be accounted for in any circuitry that makes use of BLOCK#. The difference is due to the fact that LOCK#, unlike the other 386 SX cycle definition signals, is not pipelined. The situation is clarified in Figure 5-3K. In cycle 2 the state of LOCK# is not known before the corresponding system read starts (Frame 4 and 5). In this case, LOCK# is asserted at the beginning of T1P, and the delay for BLOCK# to become active is the delay of LOCK# from the 386 SX plus the propagation delay through the 82385SX. This occurs because T1P and the corresponding BT1P are concurrent (Frame 5). The result is that BLOCK# should not be sampled at the end of BT1P. The first appropriate sampling point is midway through the next state, as shown in Frame 6. In

Figure 5-3L, the maximum delay for BLOCK# to become valid in Frame 4 is the same as the maximum delay for LOCK# to become valid from the 386 SX. This is true since the pipelining issue discussed above does not occur.

The 82385 should negate BLOCK# after BREADY# of the last 82385 Locked Cycle was asserted AND LOCK# turns inactive.

This means that in a sequence of cycles which begins with a 82385 Locked Cycle and goes on with all the possible Locked Cycles (other 82385 cycles, idles, and local cycles), while LOCK# is continuously active, the 82385 will maintain BLOCK# active continuously. Another implication is that in a Locked Posted Write Cycle followed by non-locked sequence, BLOCK# is negated one CLK after BREADY# of the write cycle. In other 82385 Locked Cycles, followed by non-locked sequences, BLOCK# is negated one CLK after LOCK# is negated, which occurs two CLKs after BREADY# is asserted. In the last case BLOCK# active moves by one CLK to the non-locked sequence.

The arbitration rules of Locked Cycles are:

MASTER MODE:

BHOLD input signal is ignored when BLOCK# or internal lock (16-bit non-aligned cycle) are active. BHLDA output signal remains inactive, and BAOE# output signal remains active at that time interval.

SLAVE MODE:

The 82385 does not relinquish the system bus if BLOCK# or internal lock are active. The BHOLD

output signal remains active when BLOCK# or internal lock is active plus one CLK. The BHLDA input signal is ignored when BLOCK# or the internal lock is active plus one CLK. This means the 82385 slave does not respond to BHLDA inactivation. The BAOE# output signal remains active during the same time interval.

5.2.2 ADDITIONAL 82385SX BUS SIGNALS

The 82385SX bus provides two status outputs and one control input that are unique to cache operation and thus have no 386 SX counterparts. The outputs are MISS# and WBS, and the input is FLUSH.

5.2.2.1 Cache Read/Write Miss Indication (MISS#)

MISS# can be thought of as an extra 82385SX bus cycle definition signal similar to BM/IO#, BW/R#, and BD/C#, that distinguishes cacheable read and write misses from other cycles. MISS#, like the other definition signals, becomes valid with BADS# (BT1 or first BT2P). The behavior of MISS# is illustrated in Figures 5-3B, 5-3C, and 5-3J. The 82385SX floats MISS# when another master owns the bus, allowing multiple 82385SXs to share the same node in multi-cache systems. MISS# should thus be lightly pulled up (~20K) to keep it negated during hold (BTH) states.

MISS# can serve several purposes. As discussed previously, the BDO-BD15 and BREADY# setup times in a cache read miss are longer than in other cycles. A bus controller can distinguish these cycles by gating MISS# with BW/R#. MISS# may also prove useful in gathering 82385SX system performance data.

5.2.2.2 WRITE BUFFER STATUS (WBS)

WBS is activated when 386 SX write cycle data is latched into the 74676 latching transceiver (via LDSTB). It is deactivated upon completion of the write cycle on the 82385SX bus when the 82385SX sees the BREADY# signal. WBS behavior is illustrated in Figures 5-3F through 5-3J, and potential applications are discussed in Chapter 3.

5.2.2.3 Cache Flush (FLUSH)

FLUSH is an 82385SX input which is used to reset all tag valid bits within the cache directory. The FLUSH input must be kept active for at least 4 CLK (8 CLK²) periods to complete the directory flush. Flush is generally used in diagnostics but can also be used in applications where snooping cannot guarantee coherency.

5.3 Bus Watching (Snoop) Interface

The 82385SX's bus watching interface consists of the snoop address (SA1-SA23), snoop strobe (SSTB#), and snoop enable (SEN) inputs. If masters reside at the system bus level, then the SA1-SA23 inputs are connected to the system address lines and SEN to the system bus memory write command. SSTB# indicates that a valid address is present on the system bus. Note that the snoop bus inputs are synchronous, so care must be taken to ensure that they are stable during their sample windows. If no master resides beyond the 82385SX bus level, then SA1-SA23, SEN and SSTB# can respectively tie directly to BA1-BA23, BW/R#, and BADS#. However, it is recommended that SEN be driven by the logical "AND" of BW/R# and BM/IO# so as to prevent I/O writes from unnecessarily invalidating cache data.

When the 82385SX detects a system write by another master, it internally latches SA1-SA23 and runs a cache look-up to see if the altered main memory location is duplicated in the cache. If yes (a snoop hit), the line valid bit associated with that cache entry is cleared. An important feature of the 82385SX is that even if the 386 SX is running zero wait state hits out of the cache, all snoops are serviced. This is accomplished by time multiplexing the cache directory between the 386 SX address and latched system address. If the SSTB# signal occurs during an 82385SX comparison cycle (for the 386 SX), the 386 SX cycle has the highest priority in accessing the cache directory. This takes the first of the two 386 SX states. The other state is then used for the snoop comparison. This worst case example, depicted in Figure 5-4, shows the 386 SX running zero wait state hits on the 386 SX local bus, and another master running zero wait state writes on the 82385SX bus. No snoops are missed, and no performance penalty incurred.

5.4 Reset Definition

Table 5-1 summarizes the states of all 82385SX outputs during reset and initialization. A slave mode 82385SX tri-states its "386 SX-like" front end. A master mode 82385SX emits a pulse stream on its BACP output. As the 386 SX address and cycle definition lines reach their reset values, this stream will latch the reset values through to the 82385SX bus.

Table 5-1. Pin State during RESET and Initialization

Output Name	Signal Level during RESET and Initialization	
	Master Mode	Slave Mode
NA#	High	High
READY0#	High	High
BRDYEN#	High	High
CALEN	High	High
CWEA# -CWEB#	High	High
CS0#, CS1#	Low	Low
CT/R#	High	High
COEA# -COEB#	High	High
BADS#	High	High Z
BBHE#, BBLE#	386 BE#	High Z
BLOCK#	High	High Z
MISS#	High	High Z
BACP	Pulse ⁽¹⁾	Pulse
BAOE#	Low	High
BT/R#	Low	Low
DOE#	High	High
LDSTB	Low	Low
BHOLD	—	Low
BHLDA	Low	—
WBS	Low	Low

NOTE:

1. In Master Mode, BAOE# is low and BACP emits a pulse stream during reset. As the 386 SX address and cycle definition signals reach their reset values, the pulse stream on BACP will latch these values through to the 82385 SX local bus.

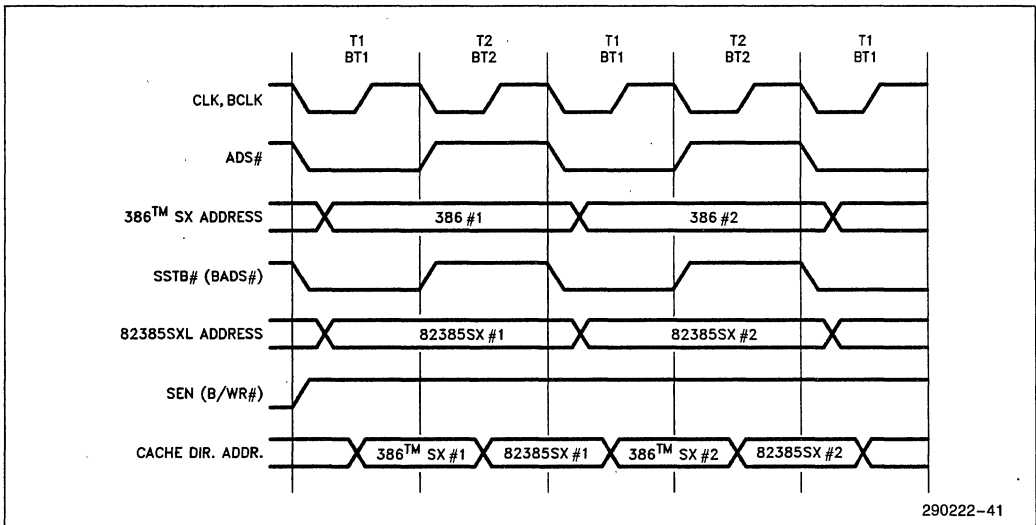


Figure 5.4. Interleaved Snoop and 80386 Accesses to the Cache Directory

6.0 82385SX SYSTEM DESIGN CONSIDERATIONS

6.1 Introduction

This chapter discusses techniques which should be implemented in an 82385SX system. Because of the high frequencies and high performance nature of the 386 SX CPU/82385SX system, good design and layout techniques are necessary. It is always recommended to perform a complete design analysis of new system designs.

6.2 Power and Grounding

6.2.1 POWER CONNECTIONS

The 82385SX utilizes 8 power (V_{CC}) and 10 ground (V_{SS}) pins. All V_{CC} and V_{SS} pins must be connected to their appropriate plane. On a printed circuit board, all V_{CC} pins must be connected to the power plane and all V_{SS} pins must be connected to the ground plane.

6.2.2 POWER DECOUPLING

Although the 82385SX itself is generally a "passive" device in that it has a few output signals, the cache subsystem as a whole is quite active. Therefore, many decoupling capacitors should be placed around the 82385SX cache subsystem.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the decoupling capacitors and their respective devices as much as possible. Capacitors specifically for PGA packages are also commercially available, for the lowest possible inductance.

6.2.3 RESISTOR RECOMMENDATIONS

Because of the dual structure of the 82385SX sub-system (386 SX Local Bus and 82385SX Local Bus), any signals which are recommended to be pulled up will be respective to one of the busses. The following sections will discuss signals for both busses.

6.2.3.1 386 SX LOCAL BUS

For typical designs, the pullup resistors shown in Table 6-1 are recommended. This table correlates to Chapter 7 of the 386 SX Data Sheet. However, particular designs may have a need to differ from the listed values. Design analysis is recommended to determine specific requirements.

6.2.3.2 82385SX Local Bus

Pullup resistor recommendations for the 82385SX Local Bus signals are shown in Table 6-2. Design analysis is necessary to determine if deviations to the typical values given are needed.

Table 6-1. Recommended Resistor Pullups to V_{CC} (386™ SX Local Bus)

Pin and Signal	Pullup Value	Purpose
ADS# PGA E13 PQFP 123	20 K Ω \pm 10%	Lightly Pull ADS# Negated for 386 SX Hold States
LOCK# PGA F13 PQFP 118	20 K Ω \pm 10%	Lightly Pull LOCK# Negated for 386 SX Hold States

Table 6-2. Recommended Resistor Pullups to V_{CC} (82385SX Local Bus)

Signal and Pin	Pullup Value	Purpose
BADS# PGA N9 PQFP 89	20 K Ω \pm 10%	Lightly Pull BADS# Negated for 82385SX Hold States
BLOCK# PGA P9 PQFP 86	20 K Ω \pm 10%	Lightly Pull BLOCK# Negated for 82385SX Hold States
MISS# PGA N8 PQFP 85	20 K Ω \pm 10%	Lightly Pull MISS# Negated for 82385SX Hold States

6.3 82385SX Signal Connections

6.3.1 CONFIGURATION INPUTS

The 82835 configuration signals (M/S#, 2W/D#, DEFOE#) must be connected (pulled up) to the appropriate logic level for the system design. There is also a reserved 82385SX input which must be tied to the appropriate level. Refer to Table 6-3 for the signals and their required logic level.

Table 6-3. 82385SX Configuration Inputs Logic Levels

Pin and Signal	Logic Level	Purpose
M/S # PGA B13 PQFP 124	High	Master Mode Operation
	Low	Slave Mode Operation
2W/D # PGA D12 PQFP 127	High	2-Way Set Associative
	Low	Direct Mapped
Reserved PGA L14 PQFP 102	High	Must be tied to V _{CC} via a pull-up for proper functionality
DEFOE # PGA A14 PQFP 128	N/A	Define Cache Output Enable. Allows use of any SRAM.

NOTE:

The listed 82385SX pins which need to be tied high should use a pull-up resistor in the range of 5 K Ω to 20 K Ω .

6.3.2 CLK2 and RESET

The 82385SX has two inputs to which the 386 SX CLK2 signal must be connected. One is labeled CLK2 (82385SX pin C13) and the other is labeled BCLK2 (82385SX pin L13). These two inputs must be tied together on the printed circuit board.

The 82385SX also has two reset inputs. RESET (82385SX pin D13) and BRESET (82385SX pin K12) must be connected on the printed circuit board.

6.4 Unused Pin Requirements

For reliable operation, ALWAYS connect unused inputs to a valid logic level. As is the case with most other CMOS processes, a floating input will increase the current consumption of the component and give an indeterminate state to the component.

6.5 Cache SRAM Requirements

The 82385SX offers the option of using SRAMs with or without an output enable pin. This is possible by inserting a transceiver between the SRAMs and the 386 SX local data bus and strapping DEFOE # to the appropriate logic level for a given system configuration. This transceiver may also be desirable in a system which has a very heavily loaded 386 SX local data bus. The following sections discuss the SRAM requirements for all cache configurations.

6.5.1 CACHE MEMORY WITHOUT TRANSCEIVERS

As discussed in Section 3.2, the 82385SX presents all of the control signals necessary to access the cache memory. The SRAM chip selects, write enables, and output enables are driven directly by the 82385SX. Table 6-4 lists the required SRAM specifications. These specifications allow for zero margins. They should be used as guides for the actual system design.

Table 6-4. SRAM Specs for Non-Buffered Cache Memory

SRAM Spec Requirements				
	Direct Mapped		2-Way Set Associative	
	16 MHz	20 MHz	16 MHz	20 MHz
Read Cycle Requirements				
Address Access (MAX)	64 ns	44 ns	62 ns	42 ns
Chip Select Access (MAX)	76	56	76	56
OE # to Data Valid (MAX)	25	19	19	14
OE # to Data Float (MAX)	20	20	20	20
Write Cycle Requirements				
Chip Select to End of Write (MIN)	40	30	40	30
Address Valid to End of Write (MIN)	58	42	56	40
Write Pulse Width (MIN)	40	30	40	30
Data Setup (MAX)	—	—	—	—
Data Hold (MIN)	4	4	4	4

6.5.2 CACHE MEMORY WITH TRANSCEIVERS

To implement an 82385SX subsystem using cache memory transceivers, COEA# or COEB# must be used as output enable signals for the transceivers and DEFOE# must be appropriately strapped for proper COEx# functionality (since the cache SRAM transceivers must be enabled for writes as well as reads). DEFOE# must be tied high when using cache SRAM transceivers. In a 2-way set associative organization, COEA# enables the transceiver for bank A and COEB# enables the bank B transceiver. A direct mapped cache may use either COEA# or COEB# to enable the transceiver. Table 6-5 lists the required SRAM specifications. These specifications allow for zero margin. They should be used as guides for the actual system design.

7.0 SYSTEM TEST CONSIDERATIONS

7.1 Introduction

Power On Self Testing (POST) is performed by most systems after a reset. This chapter discusses the requirements for properly testing an 82385SX based system after power up.

7.2 Main Memory (DRAM) Testing

Most systems perform a memory test by writing a data pattern and then reading and comparing the

data. This test may also be used to determine the total available memory within the system. Without properly taking into account the 82385SX cache memory, the memory test can give erroneous results. This will occur if the cache responds with read hits during the memory test routine.

7.2.1 MEMORY TESTING ROUTINE

In order to properly test main memory, the test routine must not read from the same block consecutively. For instance, if the test routine writes a data pattern to the first 16 Kbytes of memory (0000-3FFFH), reads from the same block, writes a new pattern to the same locations (0000-3FFFH), and read the new pattern, the second pattern tested would have had data returned from the 82385SX cache memory. Therefore, it is recommended that the test routine work with a memory block of at least 32 Kbytes. This will guarantee that no 16 Kbyte block will be read twice consecutively.

7.3 82385SX Cache Memory Testing

With the addition of SRAMs for the cache memory, it may be desirable for the system to be able to test the cache SRAMs during system diagnostics. This requires the test routine to access only the cache memory. The requirements for this routine are based on where it resides within the memory map. This can

Table 6-5. SRAM Specs for Buffered Cache Memory

SRAM Spec Requirements				
	Direct Mapped		2-Way Set Associative	
	16 MHz	20 MHz	16 MHz	20 MHz
Read Cycle Requirements				
Address Access (MAX)	57 ns	37 ns	55 ns	35 ns
Chip Select Access (MAX)	68	48	68	48
OE# to Data Valid (MAX)	N/A	N/A	N/A	N/A
OE# to Data Float (MAX)	N/A	N/A	N/A	N/A
Write Cycle Requirements				
Chip Select to End of Write (MIN)	40	30	40	30
Address Valid to End of Write (MIN)	58	42	56	40
Write Pulse Width (MIN)	40	30	40	30
Data Setup (MAX)	25	15	25	15
Data Hold (MIN)	3	3	3	3

be broken into two areas: the routine residing in cacheable memory space or the routine residing in either non-cacheable memory or on the 386 SX local bus (using the LBA# input).

7.3.1 TEST ROUTINE IN THE NCA# OR LBA# MEMORY MAP

In this configuration, the test routine will never be cached. The recommended method is code which will access a single 16 Kbyte block during the test. Initially, a 16 Kbyte read (assume 0000–3FFFH) must be executed. This will fill the cache directory with the address information which will be used in the diagnostic procedure. Then, a 16 Kbyte write to the same address locations (0000–3FFFH) will load the cache with the desired test pattern (due to write hits). The comparison can be made by completing another 16 Kbyte read (same locations, 0000–3FFFH), which will be cache read hits. Subsequent writes and reads to the same addresses will enable various patterns to be tested.

7.3.2 TEST ROUTINE IN CACHEABLE MEMORY

In this case, it must be understood that the diagnostic routine must reside in the cache memory before the actual data testing can begin. Otherwise, when the 386 SX performs a code fetch, a location within the cache memory which is to be tested will be altered due to the read miss (code fetch) update.

The first task is to load the diagnostic routine into the top of the cache memory. It must be known how much memory is required for the code as the rest of the cache memory will be tested as in the earlier method. Once the diagnostics have been cached (via read updates), the code will perform the same type of read/write/read/compare as in the routine explained in the above section. The difference is that now the amount of cache memory to be tested is 16 Kbytes minus the length of the test routine.

7.4 82385SX Cache Directory Testing

Since the 82385SX does not directly access the data bus, it is not possible to easily complete a comparison of the cache directory. (The 82385SX can serially transmit its directory contents. See Section 7.5.) However, the cache memory tests described in Section 7.3 will indicate if the directory is working properly. Otherwise, the data comparison within the diagnostics will show locations which fail.

There is a slight possibility that the cache memory comparison could pass even if locations within the directory gave false hit/miss results. This could cause the comparison to always be performed to main memory instead of the cache and give a proper

comparison to the 386 SX. The solution here is to use the MISS# output of the 82385SX as an indicator to a diagnostic port which can be read by the 386 SX. It could also be used to flag an interrupt if a failure occurs.

The implementation of these techniques in the diagnostics will assure proper functionality of the 82385SX subsystem.

7.5 Special Function Pins

As mentioned in Chapter 3, there are three 82385SX pins which have reserved functions in addition to their normal operational functions. These pins are MISS#, WBS, and FLUSH.

As discussed previously, the 82385SX performs a directory flush when the FLUSH input is held active for at least 4 CLK (8 CLK2) cycles. However, the FLUSH pin also serves as a diagnostic input to the 82385SX. The 82385SX will enter a reserved mode if the FLUSH pin is high at the falling edge of RESET.

If, during normal operation, the FLUSH input is active for only one CLK (2 CLK2) cycle/s, the 82385SX will enter another reserved mode. Therefore it must be guaranteed that FLUSH is active for at least the 4 CLK (8 CLK2) cycle specification.

WBS and MISS# serve as outputs in the 82385SX reserved modes.

8.0 MECHANICAL DATA

8.1 Introduction

This chapter discusses the physical package and its connections in detail.

8.2 Pin Assignment

The 82385SX PGA pinout as viewed from the top side of the component is shown by Figure 8-1. Its pinout as viewed from the Pin side of the component is shown in Figure 8-2.

The 82385SX Plastic Quad Flat Pack (PQFP) pinout from the top side of the component is shown by Figure 8-3.

V_{CC} and V_{SS} connections must be made to multiple V_{CC} and V_{SS} (GND) pins. Each V_{CC} and V_{SS} must be connected to the appropriate voltage level. The circuit board should include V_{CC} and GND planes for power distribution and all V_{CC} and V_{SS} pins must be connected to the appropriate plane.

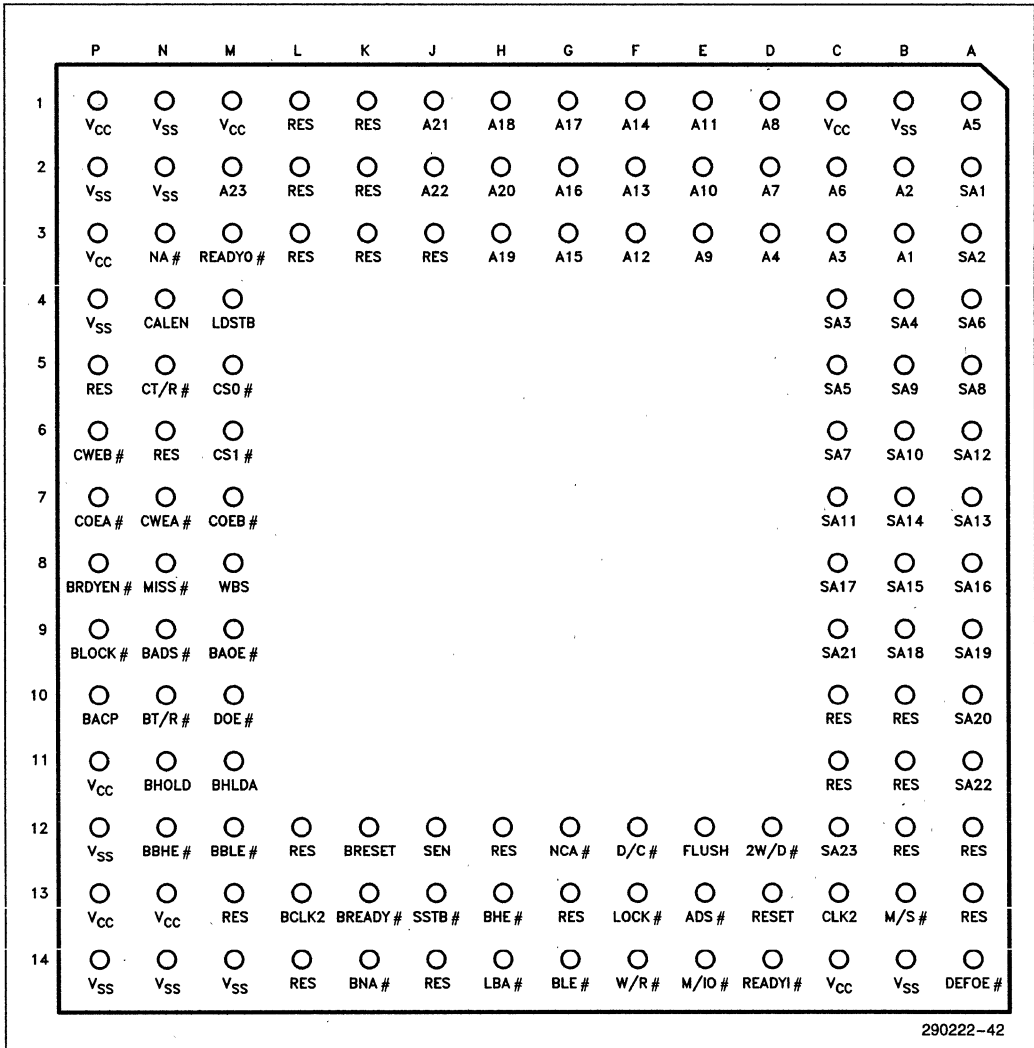


Figure 8-1. 82385SX PGA Pinout—View from TOP Side

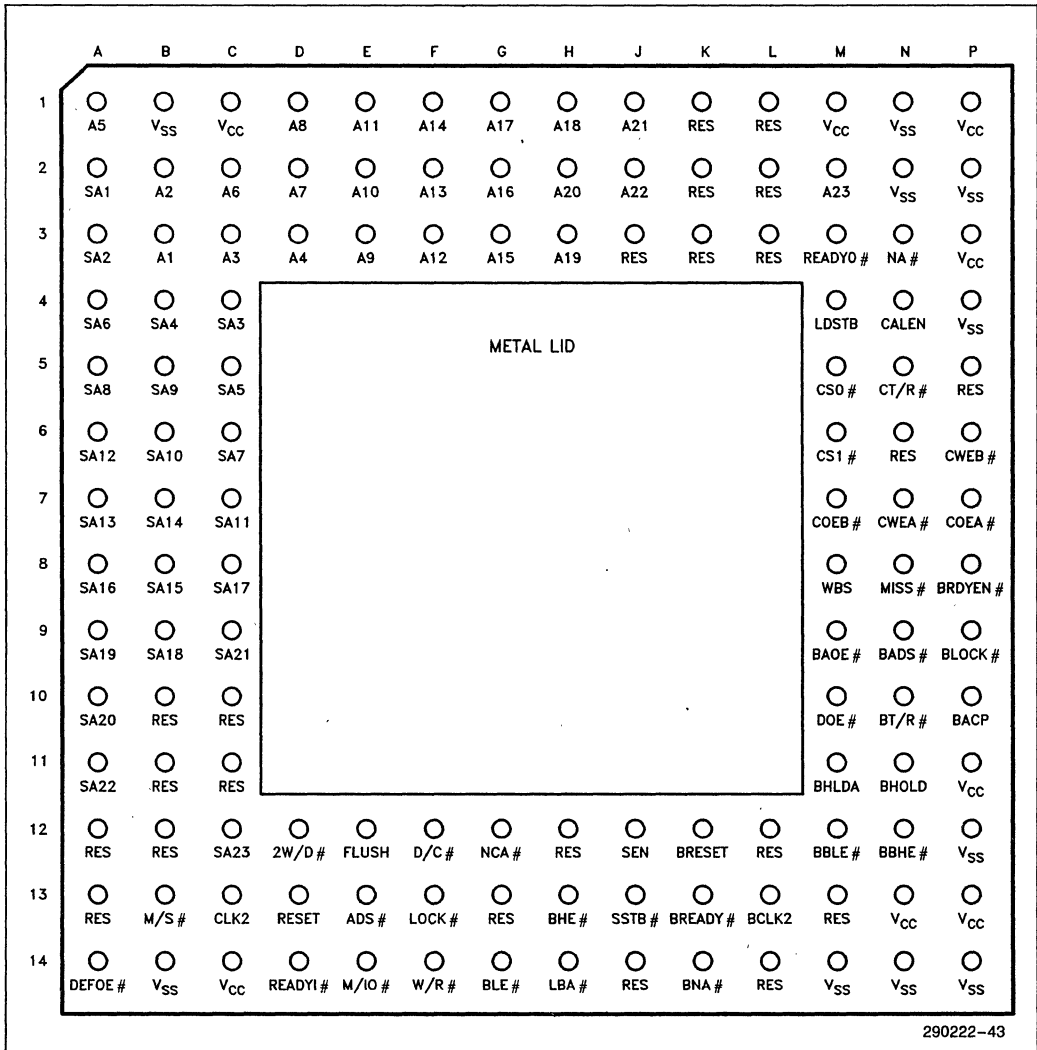


Figure 8-2. 82385SX PGA Pinout—View from PIN Side

290222-43

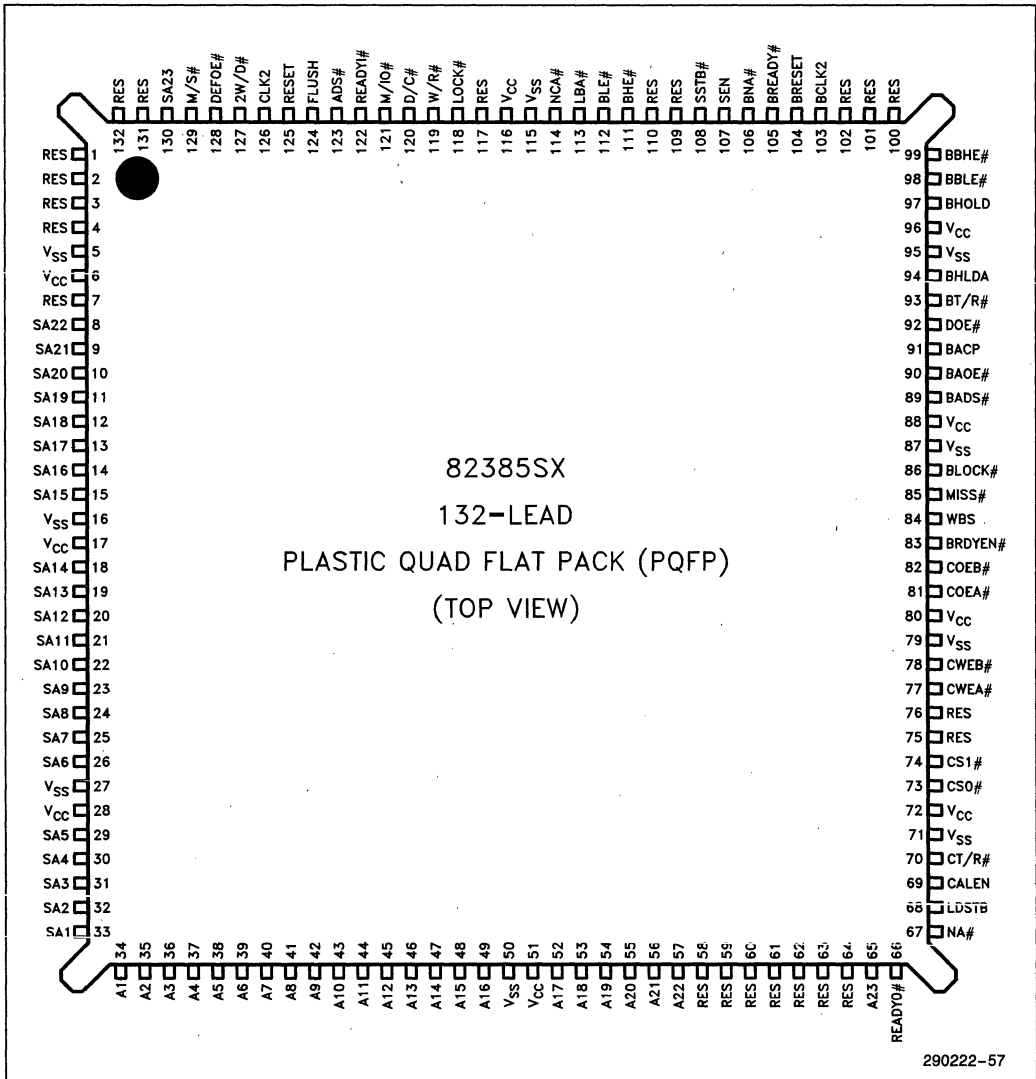


Figure 8-3. 82385SX PQFP Pinout—View from TOP Side

Table 8-1. 82385SX Pinout—Functional Grouping

PGA	PQFP	Signal	PGA	PQFP	Signal	PGA	PQFP	Signal	PGA	PQFP	Signal
M2	65	A23	G12	114	NCA#	N3	67	NA#	N5	70	CT/R#
J2	57	A22	H14	113	LBA#	E12	124	FLUSH	P8	83	BRDYEN#
J1	56	A21	D14	122	READYI#	M8	84	WBS	K13	105	BREADY#
H2	55	A20	M3	66	READYO#	N8	85	MISS#	P10	91	BACP
H3	54	A19	C12	130	SA23	A14	128	DEFOE#	M9	90	BAOE#
H1	53	A18	A11	8	SA22	B13	129	M/S#	N10	93	BT/R#
G1	52	A17	C9	9	SA21	D12	127	2W/D#	A12	1	RESERVED
G2	49	A16	A10	10	SA20	M10	92	DOE#	A13	2	RESERVED
G3	48	A15	A9	11	SA19	M4	68	LDSTB	B10	3	RESERVED
F1	47	A14	B9	12	SA18	N11	97	BHOLD	B11	4	RESERVED
F2	46	A13	C8	13	SA17	M11	94	BHLDA	B12	7	RESERVED
F3	45	A12	A8	14	SA16	B1	5	V _{SS}	C10	58	RESERVED
E1	44	A11	B8	15	SA15	B14	16	V _{SS}	C11	59	RESERVED
E2	43	A10	B7	18	SA14	M14	27	V _{SS}	G13	60	RESERVED
E3	42	A9	A7	19	SA13	N1	50	V _{SS}	H12	61	RESERVED
D1	41	A8	A6	20	SA12	N2	71	V _{SS}	J3	62	RESERVED
D2	40	A7	C7	21	SA11	N14	79	V _{SS}	J14	63	RESERVED
C2	39	A6	B6	22	SA10	P2	87	V _{SS}	K1	64	RESERVED
A1	38	A5	B5	23	SA9	P4	95	V _{SS}	K2	75	RESERVED
D3	37	A4	A5	24	SA8	P12	115	V _{SS}	K3	76	RESERVED
C3	36	A3	C6	25	SA7	P14	—	V _{SS}	L1	100	RESERVED
B2	35	A2	A4	26	SA6	N9	89	BADS#	L2	101	RESERVED
B3	34	A1	C5	29	SA5	M12	98	BBLE#	L3	102	RESERVED
G14	112	BLE#	B4	30	SA4	N12	99	BBHE#	L12	109	RESERVED
H13	111	BHE#	C4	31	SA3	P9	86	BLOCK#	L14	110	RESERVED
C13	126	CLK2	A3	32	SA2	K14	106	BNA#	M13	117	RESERVED
D13	125	RESET	A2	33	SA1	N4	69	CALEN	N6	131	RESERVED
K12	104	BRESET	J12	107	SEN#	P7	81	COEA#	P5	132	RESERVED
L13	103	BCLK2	J13	108	SSTB#	M7	82	COEB#			
F14	119	W/R#	C1	6	V _{CC}	N7	77	CWEA#			
F12	120	D/C#	C14	17	V _{CC}	P6	78	CWEB#			
E14	121	M/IO#	M1	28	V _{CC}	M5	73	CS0#			
F13	118	LOCK#	N13	51	V _{CC}	M6	74	CS1#			
F13	118	LOCK#	P1	72	V _{CC}						
E13	123	ADS#	P3	80	V _{CC}						
			P11	88	V _{CC}						
			P13	96	V _{CC}						
			—	116	V _{CC}						

8.3 Package Dimensions and Mounting

The 82385SX PGA package is a 132-pin ceramic Pin Grid Array. The pins are arranged 0.100 inch (2.54 mm) center-to-center, in a 14 × 14 matrix, three rows around.

A wide variety of available PGA sockets allow low insertion force or zero insertion force mounting.

These come in a choice of terminals such as solder-tail, surface mount, or wire wrap.

The 82385SX PQFP is a 132-lead Plastic Quad Flat Pack. The pins are "fine pitch", 0.025 inches (0.635 mm) center to center.

The PQFP device is intended to be surface mounted directly to the printed board although sockets are available for this device.

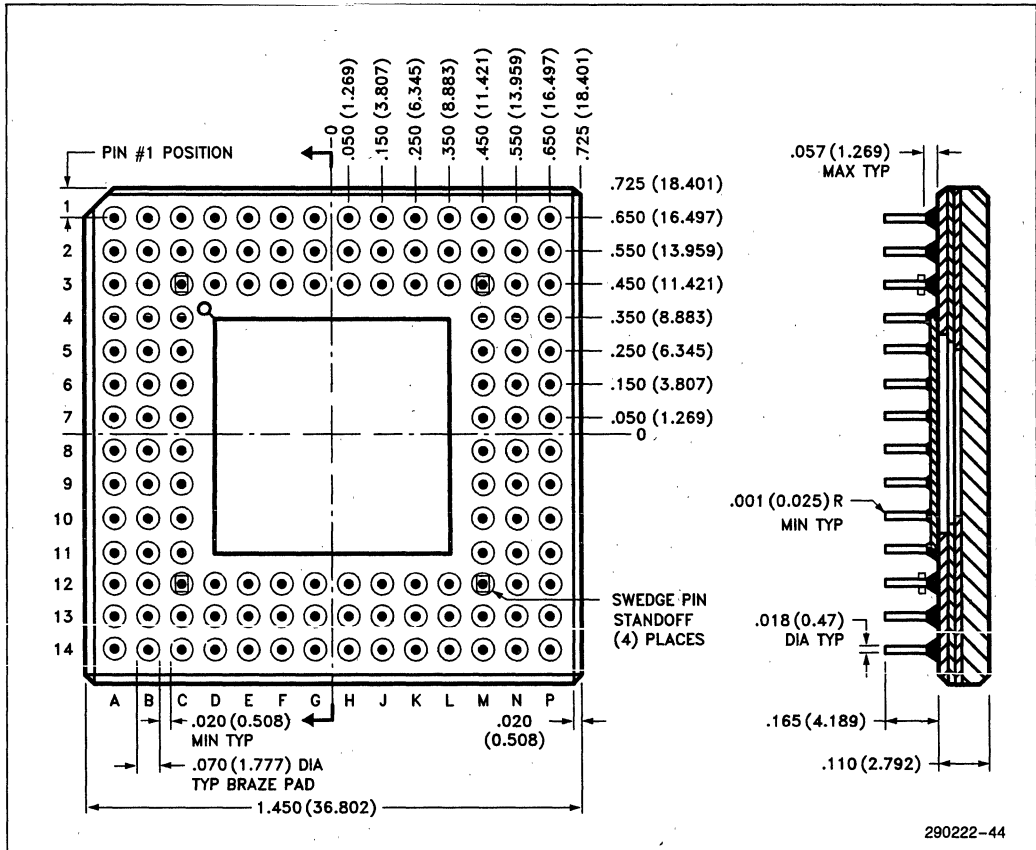


Figure 8-3.1. 132-Pin PGA Package Dimensions

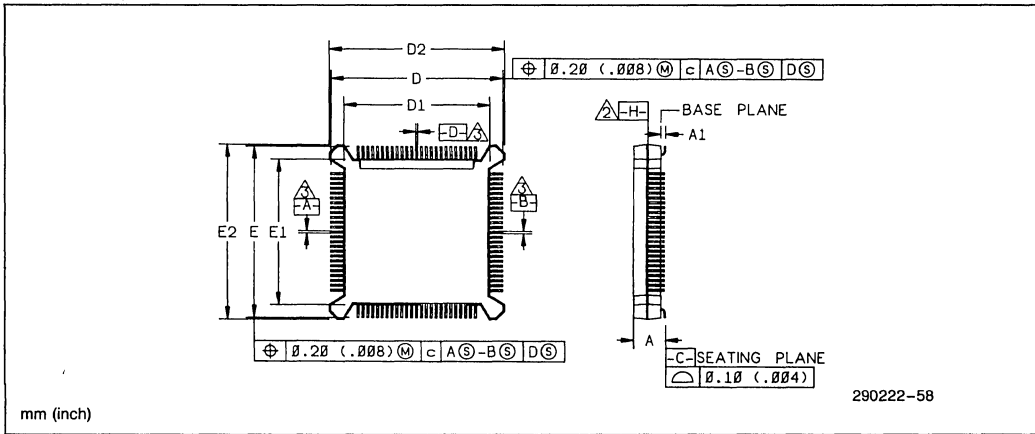


Figure 8-3.2. Principal Dimensions and Datums

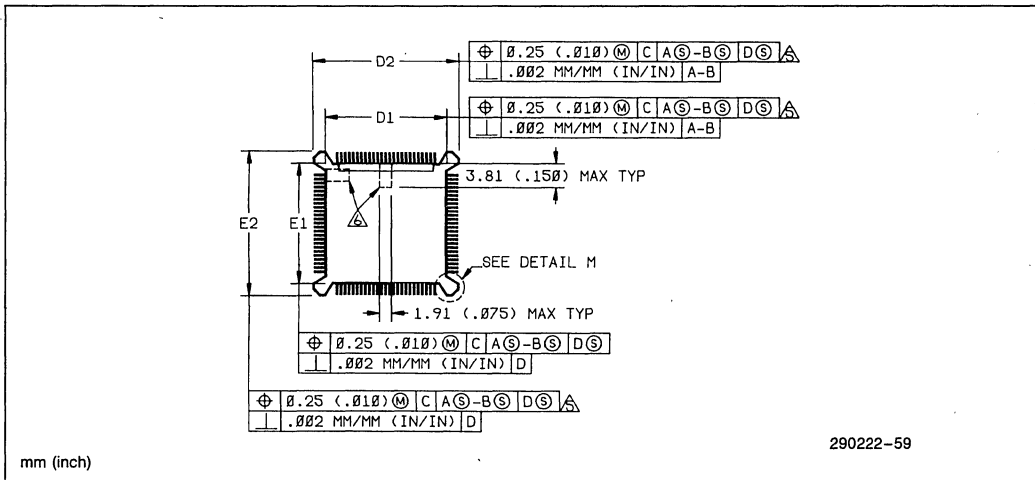


Figure 8-3.3. Molded Details

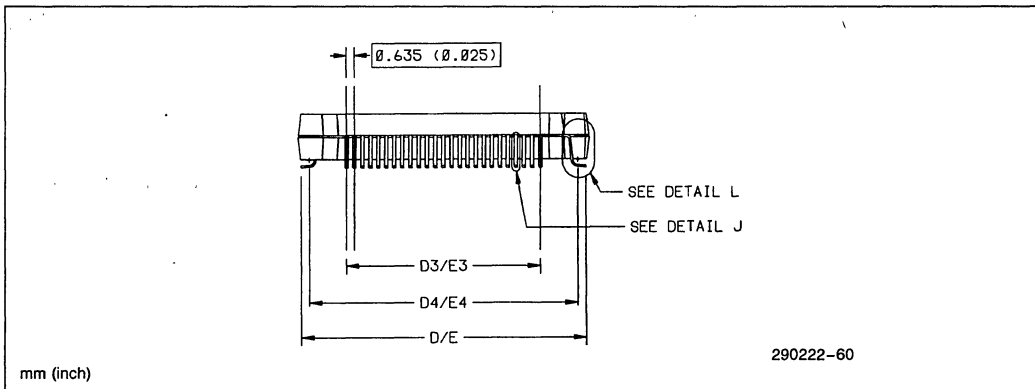


Figure 8-3.4. Terminal Details

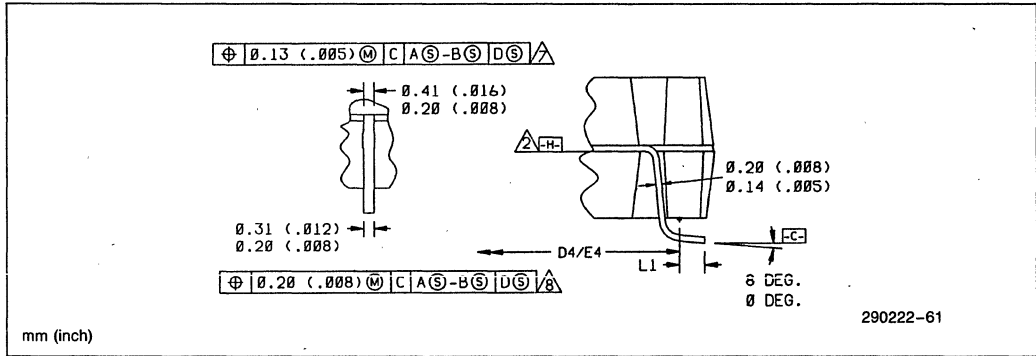


Figure 8-3.5. Typical Lead

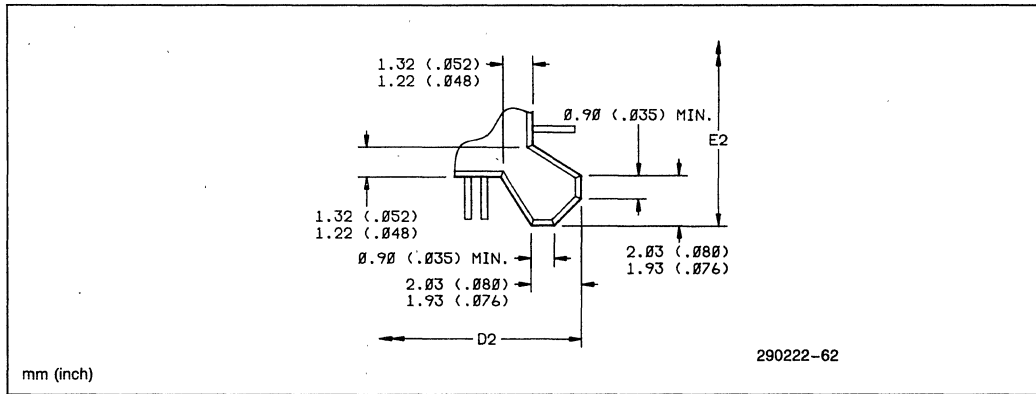


Figure 8-3.6. Detail M

PLASTIC QUAD FLAT PACK

Table 8-3.1. Symbol List for Plastic Quad Flat Pack

Letter or Symbol	Description of Dimensions
A	Package height: distance from seating plane to highest point of body
A1	Standoff: Distance from seating plane to base plane
D/E	Overall package dimension: lead tip to lead tip
D1/E1	Plastic body dimension
D2/E2	Bumper Distance
D3/E3	Footprint
L1	Foot length
N	Total number of leads

NOTES:

- All dimensions and tolerances conform to ANSI Y14.5M-1982.
- Datum plane -H- located at the mold parting line and coincident with the bottom of the lead where lead exits plastic body.
- Datums A-B and -D- to be determined where center leads exit plastic body at datum plane -H-.
- Controlling Dimension, Inch.
- Dimensions D1, D2, E1 and E2 are measured at the mold parting line and do not include mold protrusion. Allowable mold protrusion of 0.18 mm (0.007 in) per side.
- Pin 1 identifier is located within one of the two zones indicated.
- Measured at datum plane -H-.
- Measured at seating plane datum -C-.

Table 8-3.2. PQFP Dimensions and Tolerances

Intel Case Outline Drawings Plastic Quad Flat Pack 0.025 Inch Pitch				Intel Case Outline Drawings Plastic Quad Flat Pack 0.64 mm Pitch			
Symbol	Description	Min	Max	Symbol	Description	Min	Max
N	Leadcount	132		N	Leadcount	132	
A	Package Height	0.160	0.170	A	Package Height	4.06	4.32
A1	Standoff	0.020	0.030	A1	Standoff	0.51	0.76
D, E	Terminal Dimension	1.075	1.085	D, E	Terminal Dimension	27.31	27.56
D1, E1	Package Body	0.947	0.953	D1, E1	Package Body	24.05	24.21
D2, E2	Bumper Distance	1.097	1.103	D2, E2	Bumper Distance	27.86	28.02
D3, E3	Lead Dimension	0.800 REF		D3, E3	Lead Dimension	20.32 REF	
L1	Foot Length	0.020	0.030	L1	Foot Length	0.51	0.76
Issue	IWS Preliminary 1/15/87			Issue	IWS Preliminary 1/15/87		

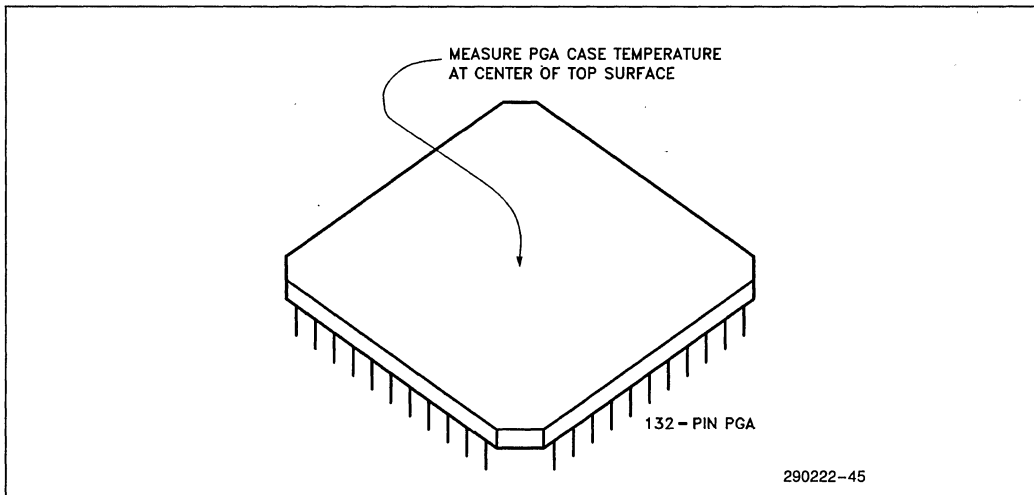


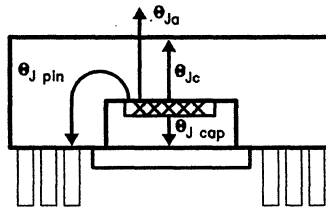
Figure 8-3.7. Measuring 82385SX PGA Case Temperature

Table 8-3.3. 82385SX PGA Package Typical Thermal Characteristics

Parameter	Thermal Resistance—°C/Watt						
	Airflow—f ³ /min (m ³ /sec)						
	0 (0)	50 (0.25)	100 (0.50)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
θ Junction-to-Case (Case Measured as Figure 8-3.7)	2	2	2	2	2	2	2
θ Case-to-Ambient (No Heatsink)	19	18	17	15	12	10	9
θ Case-to-Ambient (with Omnidirectional Heatsink)	16	15	14	12	9	7	6
θ Case-to-Ambient (with Unidirectional Heatsink)	15	14	13	11	8	6	5

NOTES:

1. Table 8-3.4 applies to 82385SX PGA plugged into socket or soldered directly onto board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{J-CAP} = 4^\circ\text{C/W}$ (approx.)
 $\theta_{J-PIN} = 4^\circ\text{C/W}$ (inner pins) (approx.)
 $\theta_{J-PIN} = 8^\circ\text{C/W}$ (outer pins) (approx.)



290222-46

Table 8-3.3. 82385 PQFP Package Typical Thermal Characteristics

Parameter	Thermal Resistance—°C/Watt						
	Airflow—f ³ /min (m ³ /sec)						
	0 (0)	50 (0.25)	100 (0.50)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
θ Junction-to-Case (Case Measured as Figure 8-3.7)							
θ Case-to-Ambient (No Heatsink)							
θ Case-to-Ambient (with Omnidirectional Heatsink)							
θ Case-to-Ambient (with Unidirectional Heatsink)							

NOTES:

1. Table 8-3.3 applies to 82385SX PQFP plugged into socket or soldered directly onto board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{J-CAP} = 4^\circ\text{C/W}$ (approx.)
 $\theta_{J-PIN} = 4^\circ\text{C/W}$ (inner pins) (approx.)
 $\theta_{J-PIN} = 8^\circ\text{C/W}$ (outer pins) (approx.)

8.4 Package Thermal Specification

The case temperature should be measured at the center of the top surface as in Figure 8-3.7 for PGA or Table 8-3.3 for PQFP. The case temperature may be measured in any environment to determine whether or not the 82385SX is within the specified operating range.

Supply Voltage
with Respect to V_{SS} -0.5V to +6.5V
Voltage on Any Other Pin -0.5V to $V_{CC} + 0.5V$

NOTE:

Stress above those listed may cause permanent damage to the device. This is a stress rating only and functional operation at these or any other conditions above those listed in the operational sections of this specification is not implied.

9.0 ELECTRICAL DATA

9.1 Introduction

This chapter presents the A.C. and D.C specifications for the 82385SX.

Exposure to absolute maximum rating conditions for extended periods may affect device reliability. Although the 82385SX contains protective circuitry to resist damage from static electric discharges, always take precautions against high static voltages or electric fields.

9.2 Maximum Ratings

Storage Temperature -65°C to +150°C
Case Temperature under Bias ... -65°C to +110°C

9.3 D.C. Specifications $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$; $V_{CC} = 5V \pm 5\%$; $V_{SS} = 0V$

Table 9-1. D.C. Specifications (16 MHz and 20 MHz)

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{IL}	Input Low Voltage	-0.3	0.8	V	(Noe 1)
V_{IH}	Input High Voltage	2.0	$V_{CC} + 0.3$	V	
V_{CL}	CLK2, BCLK2 Input Low	-0.3	0.8	V	(Note 1)
V_{CH}	CLK2, BCLK2 Input High	$V_{CC} - 0.8$	$V_{CC} + 0.3$	V	
V_{OL}	Output Low Voltage		0.45	V	$I_{OL} = 4$ mA
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -1$ mA
I_{CC}	Power Supply Current		275	mA	(Note 2)
I_{LI}	Input Leakage Current		± 15	μA	$0V < V_{IN} < V_{CC}$
I_{LO}	Output Leakage Current		± 15	μA	$0.45V < V_{OUT} < V_{CC}$
C_{IN}	Input Capacitance		10	pF	(Note 3)
C_{CLK}	CLK2 Input Capacitance		20	pF	(Note 3)

NOTES:

1. Minimum value is not 100% tested.
2. I_{CC} is specified with inputs driven to CMOS levels. I_{CC} may be higher if driven to TTL levels.
3. Sampled only.

9.4 A.C. Specifications

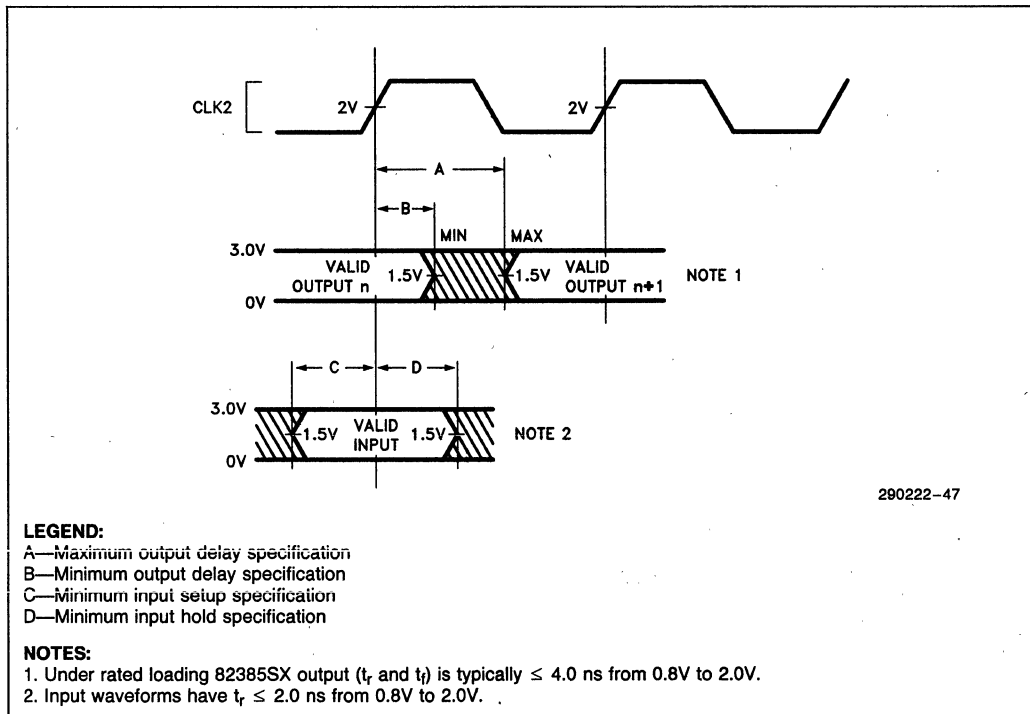
The A.C. specifications given in the following tables consist of output delays and input setup requirements. The A.C. diagram's purpose is to illustrate the clock edges from which the timing parameters are measured. The reader should not infer any other timing relationships from them. For specific information on timing relationships between signals, refer to the appropriate functional section.

A.C. spec measurement is defined in Figure 9-1. Inputs must be driven to the levels shown when A.C. specifications are measured. 82385SX output delays

are specified with minimum and maximum limits, which are measured as shown. 82385SX input setup and hold times are specified as minimums and define the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 82385SX operation.

9.4.1 FREQUENCY DEPENDENT SIGNALS

The 82385SX has signals whose output valid delays are dependent on the clock frequency. These signals are marked in the A.C. Specification Tables with a Note 1.



290222-47

LEGEND:

- A—Maximum output delay specification
- B—Minimum output delay specification
- C—Minimum input setup specification
- D—Minimum input hold specification

NOTES:

1. Under rated loading 82385SX output (t_r and t_f) is typically ≤ 4.0 ns from 0.8V to 2.0V.
2. Input waveforms have $t_r \leq 2.0$ ns from 0.8V to 2.0V.

Figure 9-1. Drive Levels and Measurement Points for A.C. Specification

A.C. SPECIFICATION TABLES

 Functional operating range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$
Table 4.1. A.C. Specifications at 16 MHz

Symbol	Parameter	Min	Max	Units	Notes
t1	Operating Frequency	15.4	16	MHz	
t2	CLK2, BCLK2 Period	31.25	32.5	ns	
t3a	CLK2, BCLK2 High Time @ 2V	10		ns	
t3b	CLK2, BCLK2 High Time @ 3.7V	7		ns	3
t4a	CLK2, BCLK2 Low Time @ 2V	10		ns	
t4b	CLK2, BCLK2 Low Time @ 0.8V	7		ns	3
t5	CLK2, BCLK2 Fall Time		8	ns	3, 9
t6	CLK2, BCLK2 Rise Time		8	ns	3, 9
t7a	A4–A12 Setup Time	30		ns	1
t7b	LOCK# Setup Time	19		ns	1
t7c	BLE#, BHE# Setup Time	21		ns	1
t7d	A1–A3, A13–A23 Setup Time	23		ns	1
t8	A1–A23, BLE#, BHE#, LOCK# Hold	3		ns	
t9a	M/IO#, D/C# Setup Time	30		ns	1
t9b	W/R# Setup Time	30		ns	1
t9c	ADS# Setup Time	30		ns	1
t10	M/IO#, D/C#, W/R#, ADS# Hold Time	5		ns	
t11	READYI# Setup Time	19		ns	1
t12	READYI# Hold Time	4		ns	
t13a1	NCA# Setup Time (See t55b2)	27		ns	6
t13a2	NCA# Setup Time (See t55b3)	20		ns	6
t13b	LBA# Setup Time	16		ns	
t14a	NCA# Hold Time	4		ns	
t14b	LBA# Hold Time	4		ns	
t15	RESET, BRESET Setup Time	13		ns	
t16	RESET, BRESET Hold Time	4		ns	
t17	NA# Valid Delay	12	42	ns	1 (25 pF Load)
t18	READYO# Valid Delay	3	31	ns	1 (25 pF Load)
t19	BRDYEN# Valid Delay	3	31	ns	
t21a1	CALEN Rising, PHI1	3	30	ns	
t21a2	CALEN Falling, PHI1	3	30	ns	
t21a3	CALEN Falling in T1P, PHI2	3	30	ns	
t21b	CALEN Rising Following CWTH	3	39	ns	1
t21c	CALEN Pulse Width	10		ns	
t21d	CALEN Rising to CS# Falling	13		ns	

A.C. SPECIFICATION TABLES (Continued)

 Functional operating range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$
Table 4.1. A.C. Specifications at 16 MHz (Continued)

Symbol	Parameter	Min	Max	Units	Notes
t22a1	CWEx# Falling, PHI1 (CWTH)	4	31	ns	1
t22a2	CWEx# Falling, PHI2 (CRDM)	4	31	ns	1
t22b	CWEx# Pulse Width	40		ns	1, 2
t22c1	CWEx# Rising, PHI1 (CWTH)	4	31	ns	1
t22c2	CWEx# Rising, PHI2 (CRDM)	4	31	ns	1
t23a1	CS1#, CS2# Rising, PHI1 (CRDM)	6	41	ns	1
t23a2	CS1#, CS2# Rising, PHI2 (CWTH)	6	41	ns	1
t23a3	CS1#, CS2# Falling, PHI1 (CWTH)	6	41	ns	1
t23a4	CS1#, CS2# Falling, PHI2 (CRDM)	6	41	ns	1
t24a1	CT/R# Rising, PHI2 (CRDH)	6	43	ns	1
t24a2	CT/R# Falling, PHI1 (CRDH)	6	43	ns	1
t24a3	CT/R# Falling, PHI2 (CRDH)	6	43	ns	1
t25a	COEA#, COEB# Falling (Direct)	4	33	ns	(25 pF Load)
t25b	COEA#, COEB# Falling (2-Way)	4	34	ns	1 (25 pF Load)
t25c1	COEx# Rising Delay @ $T_{CASE} = 0C$	4	20	ns	(25 pF Load)
t25c2	COEx# Rising Delay @ $T_{CASE} = T_{MAX}$	4	20	ns	(25 pF Load)
t23b	COEx# Falling to CSx# Rising	0		ns	
t25d	CWEx# Falling to COEx# Falling or CWEx# Rising to COEx# Rising	0	10	ns	(25 pF Load)
t26	CS0#, CS1# Falling to CWEx# Rising	40		ns	1, 2
t27	CWEx# Falling to CS0#, CS1# Falling	0		ns	
t28a	CWEx# Rising to CALEN Rising	0		ns	
t28b	CWEx# Rising to CS0#, CS1# Falling	0		ns	
t31	SA(1-23) Setup Time	25		ns	
t32	SA(1-23) Hold Time	3		ns	
t33	BADS# Valid Delay	4	33	ns	1
t34	BADS# Float Delay	4	33	ns	3
t35	BNA# Setup Time	11		ns	
t36	BNA# Hold Time	15		ns	
t37	BREADY# Setup Time	31		ns	1
t38	BREADY# Hold Time	4		ns	
t40a	BACP Rising Delay	0	26	ns	
t40b	BACP Falling Delay	0	28	ns	
t41	BAOE# Valid Delay	3	23	ns	

A.C. SPECIFICATION TABLES (Continued)Functional operating range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$ **Table 4.1. A.C. Specifications at 16 MHz** (Continued)

Symbol	Parameter	Min	Max	Units	Notes
t43a	BT/R# Valid Delay	2	27	ns	
t43b1	DOE# Falling Delay	2	30	ns	
t43b2	DOE# Rising Delay @ $T_{CASE} = 0C$	3	23	ns	
t43b3	DOE# Rising Delay @ $T_{CASE} = T_{MAX}$	3	26	ns	
t43c	LDSTB Valid Delay	2	33	ns	
t44a	SEN Setup Time	15		ns	
t44b	SSTB# Setup Time	15		ns	
t45	SEN, SSTB# Hold Time	5		ns	
t46	BHOLD Setup Time	26		ns	
t47	BHOLD Hold Time	5		ns	
t48	BHLDA Valid Delay	3	33	ns	
t55a	BLOCK# Valid Delay	3	36	ns	1, 5
t55b1	BBxE# Valid Delay	3	36	ns	1, 7
t55b2	BBxE# Valid Delay	3	36	ns	1, 7
t55b3	BBxE# Valid Delay	3	43	ns	1, 7
t55c	LOCK# Falling to BLOCK# Falling	0	36	ns	1, 5
t56	MISS# Valid Delay	3	43	ns	1
t57	MISS#, BBxE#, BLOCK# Float Delay	4	40	ns	3
t58	WBS Valid Delay	3	39	ns	1
t59	FLUSH Setup Time	21		ns	
t60	FLUSH Hold Time	5		ns	
t61	FLUSH Setup to RESET Low	31		ns	
t62	FLUSH Hold from RESET Low	31		ns	

A.C. SPECIFICATION TABLES

 Functional operating range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$
A.C. Specifications at 20 MHz

Symbol	Parameter	Min	Max	Units	Notes
t1	Operating Frequency	15.4	20	MHz	
t2	CLK2, BCLK2 Period	25	32.5	ns	
t3a	CLK2, BCLK2 High Time @ 2V	10		ns	
t3b	CLK2, BCLK2 High Time @ 3.7V	7		ns	3
t4a	CLK2, BCLK2 Low Time @ 2V	10		ns	
t4b	CLK2, BCLK2 Low Time @ 0.8V	7		ns	3
t5	CLK2, BCLK2 Fall Time		8	ns	3, 9
t6	CLK2, BCLK2 Rise Time		8	ns	3, 9
t7a1	A4–A12 Setup Time	20		ns	1
t7a2	A1–A3, A13–A19, A21–A23 Setup Time	18		ns	1
t7a3	A20 Setup Time	16		ns	1
t7b	LOCK# Setup Time	16		ns	1
t7c	BLE#, BHE# Setup Time	18		ns	1
t8	A1–A23, BLE#, BHE#, LOCK# Hold	3		ns	
t9a	M/IO#, D/C# Setup Time	20		ns	1
t9b	W/R# Setup Time	20		ns	1
t9c	ADS# Setup Time	22		ns	1
t10	M/IO#, D/C#, W/R#, ADS# Hold Time	5		ns	
t11	READYI# Setup Time	12		ns	1
t12	READYI# Hold Time	4		ns	
t13a1	NCA# Setup Time (See t55b2)	21		ns	6
t13a2	NCA# Setup Time (See t55b3)	16		ns	6
t13b	LBA# Setup Time	10		ns	
t14a	NCA# Hold Time	4		ns	
t14b	LBA# Hold Time	4		ns	
t15	RESET, BRESET Setup Time	12		ns	
t16	RESET, BRESET Hold Time	4		ns	
t17	NA# Valid Delay	12	34	ns	1 (25 pF Load)
t18	READYO# Valid Delay	3	26	ns	1 (25 pF Load)
t19	BRDYEN# Valid Delay	3	26	ns	
t21a1	CALEN Rising, PHI1	3	24	ns	
t21a2	CALEN Falling, PHI1	3	24	ns	
t21a3	CALEN Falling in T1P, PHI2	3	24	ns	
t21b	CALEN Rising Following CWTH	3	34	ns	1
t21c	CALEN Pulse Width	10		ns	

A.C. SPECIFICATION TABLES (Continued)Functional operating range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$ **A.C. Specifications at 20 MHz** (Continued)

Symbol	Parameter	Min	Max	Units	Notes
t21d	CALEN Rising to CS# Falling	13		ns	
t22a1	CWEx# Falling, PHI1 (CWTH)	4	27	ns	1
t22a2	CWEx# Falling, PHI2 (CRDM)	4	27	ns	1
t22b	CWEx# Pulse Width	30		ns	1, 2
t22c1	CWEx# Rising, PHI1 (CWTH)	4	27	ns	1
t22c2	CWEx# Rising, PHI2 (CRDM)	4	27	ns	1
t23a1	CS1#, CS2# Rising, PHI1 (CRDM)	6	37	ns	1
t23a2	CS1#, CS2# Rising, PHI2 (CWTH)	6	37	ns	1
t23a3	CS1#, CS2# Falling, PHI1 (CWTH)	6	37	ns	1
t23a4	CS1#, CS2# Falling, PHI2 (CRDM)	6	37	ns	1
t24a1	CT/R# Rising, PHI2 (CRDH)	6	38	ns	1
t24a2	CT/R# Falling, PHI1 (CRDH)	6	38	ns	1
t24a3	CT/R# Falling, PHI2 (CRDH)	6	38	ns	1
t25a	COEA#, COEB# Falling (Direct)	4	22	ns	(25 pF Load)
t25b	COEA#, COEB# Falling (2-Way)	4	24.5	ns	1 (25 pF Load)
t25c	COEx# Rising Delay	5	17	ns	(25 pF Load)
CACHE SRAM WRITE CYCLES					
t23b	COEx# Falling to CSx# Rising	0		ns	8
t25d	CWEx# Falling to COEx# Falling or CWEx# Rising to COEx# Rising	0	10	ns	8 (25 pF Load)
t26	CS0#, CS1# Falling to CWEx# Rising	30		ns	1, 2
t27	CWEx# Falling to CS0#, CS1# Falling	0		ns	
t28a	CWEx# Rising to CALEN Rising	0		ns	
t28b	CWEx# Rising to CS0#, CS1# Falling	0		ns	
t31	SA(1-23) Setup Time	19		ns	
t32	SA(1-23) Hold Time	3		ns	
t33	BADS# Valid Delay	4	28	ns	1
t34	BADS# Float Delay	4	30	ns	3
t35	BNA# Setup Time	9		ns	
t36	BNA# Hold time	15		ns	
t37	BREADY# Setup Time	26		ns	1
t38	BREADY# Hold Time	4		ns	
t40a	BACP Rising Delay	0	20	ns	
t40b	BACP Falling Delay	0	22	ns	

A.C. SPECIFICATION TABLES (Continued)

Functional operating range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$

A.C. Specifications at 20 MHz (Continued)

Symbol	Parameter	Min	Max	Units	Notes
t41	BAOE # Valid Delay	3	18	ns	
t43a	BT/R # Valid Delay	2	19	ns	
t43b1	DOE # Falling Delay	2	23	ns	
t43b2	DOE # Rising Delay @ $T_{CASE} = 0C$	4	17	ns	
t43b3	DOE # Rising Delay @ $T_{CASE} = T_{MAX}$	4	19	ns	
t43c	LDSTB Valid Delay	2	26	ns	
t44a	SEN Setup Time	11		ns	
t44b	SSTB # Setup Time	11		ns	
t45	SEN, SSTB # Hold Time	5		ns	
t46	BHOLD Setup Time	17		ns	
t47	BHOLD Hold Time	5		ns	
t48	BHLDA Valid Delay	3	28	ns	
t55a	BLOCK # Valid Delay	3	30	ns	1, 5
t55b1	BBxE # Valid Delay	3	30	ns	1, 7
t55b2	BBxE # Valid Delay	3	30	ns	1, 7
t55b3	BBxE # Valid Delay	3	36	ns	1, 7
t55c	LOCK # Falling to BLOCK # Falling	0	30	ns	1, 5
t56	MISS # Valid Delay	3	35	ns	1
t57	MISS #, BBxE #, BLOCK # Float Delay	4	32	ns	3
t58	WBS Valid Delay	3	37	ns	1
t59	FLUSH Setup Time	16		ns	
t60	FLUSH Hold Time	5		ns	
t61	FLUSH Setup to RESET Low	26		ns	
t62	FLUSH Hold from RESET Low	26		ns	

82385SX A.C. Specification Notes:

1. Frequency dependent specifications.
2. Used for cache data memory (SRAM) specifications.
3. This parameter is sampled, not 100% tested. Guaranteed by design.
5. BLOCK # delay is either from BPH1 or from 386 LOCK #. Refer to Figures 5-3K and 5-3L in the 82385SX data sheet.
6. NCA # setup time is now specified to the rising edge of BPH2 in the state after 386 SX addresses become valid (either the state after the first T2 or after the first T2P).
7. BBxE # Valid Delay is a function of NCA # setup.
 BBxE # valid delay:
 t55b1 For cacheable system bus accesses
 t55b2 For NCA # setup < t13a1
 t55b3 For t13a2 < NCA # setup < t13a1
8. t23b and t25d are only valid specifications when DEFOE # = V_{CC} . Otherwise, if DEFOE # = V_{SS} , COEx # is never asserted during cache SRAM write cycles. If DEFOE # = V_{SS} , t23b and t25d are Not Applicable.
9. t5 is measured from 0.8V to 3.7V. t6 is measured from 3.7V to 0.8V.

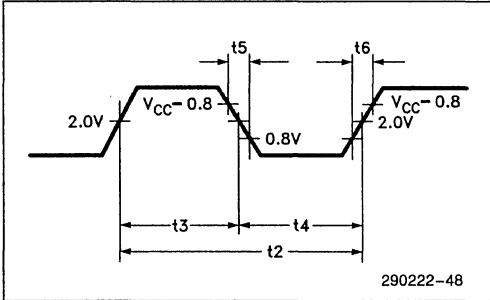


Figure 9-2. CLK2, BCLK2 Timing

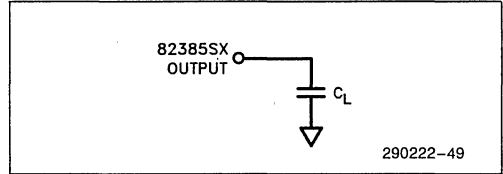
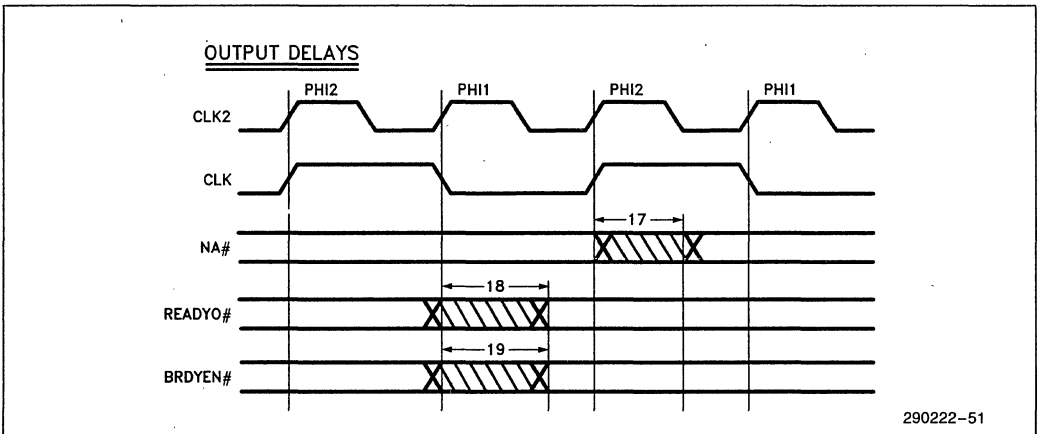
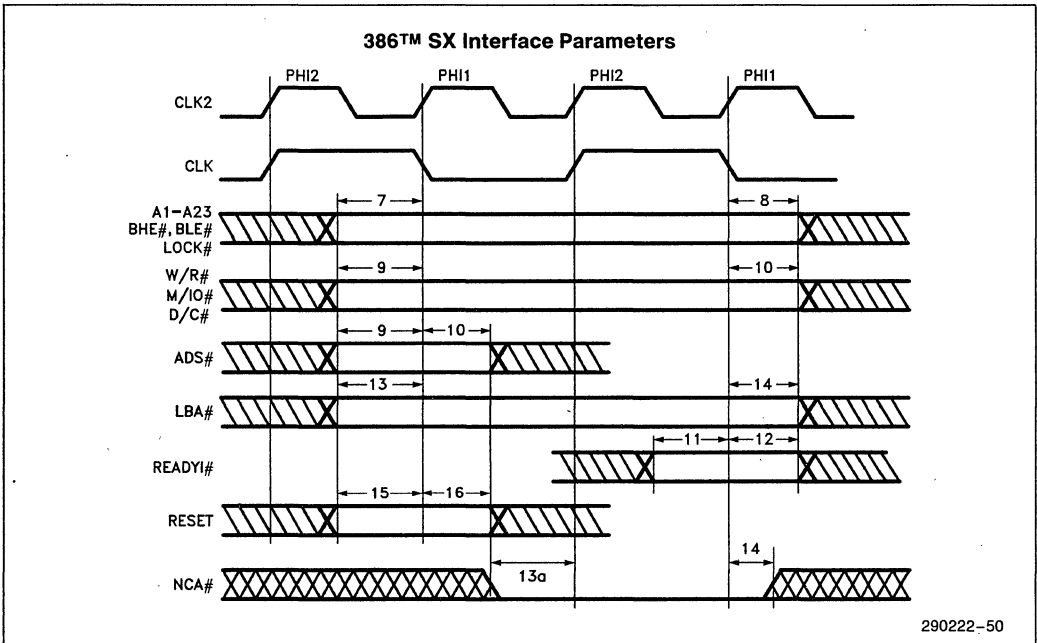
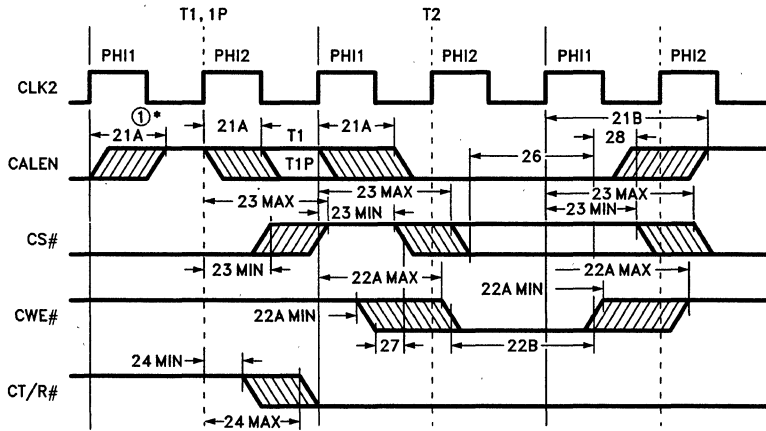


Figure 9-3. A.C. Test Load



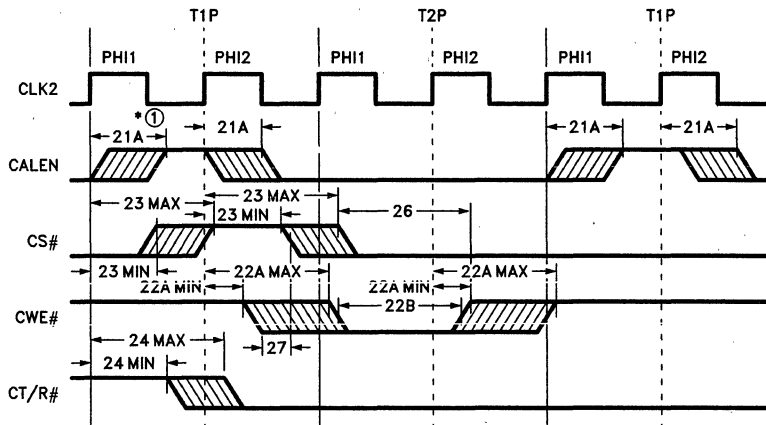
Cache Write Hit Cycle



①*. This would be 21B if previous bus cycle was Cache Write Hit cycle.

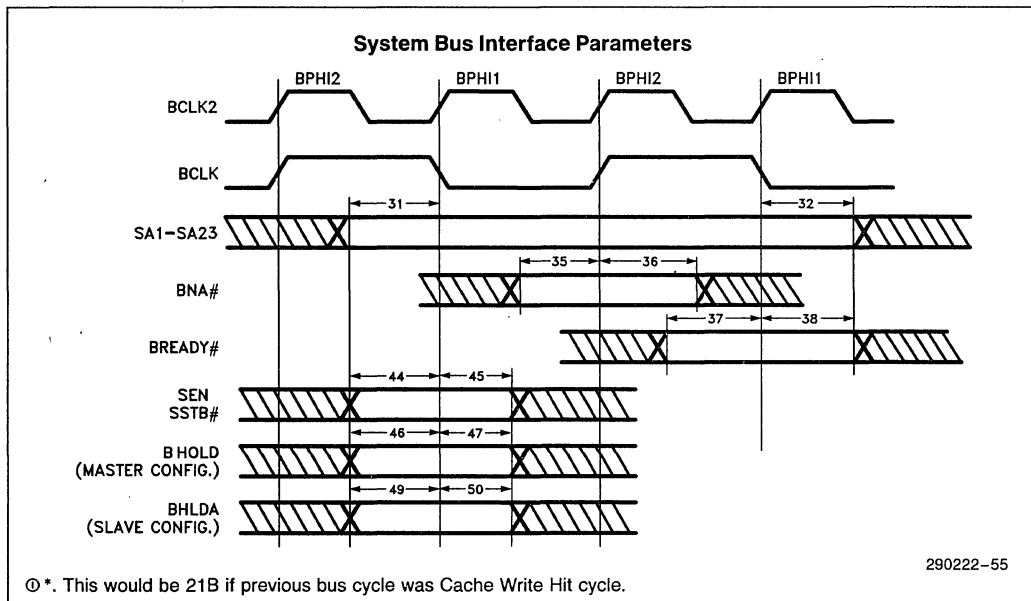
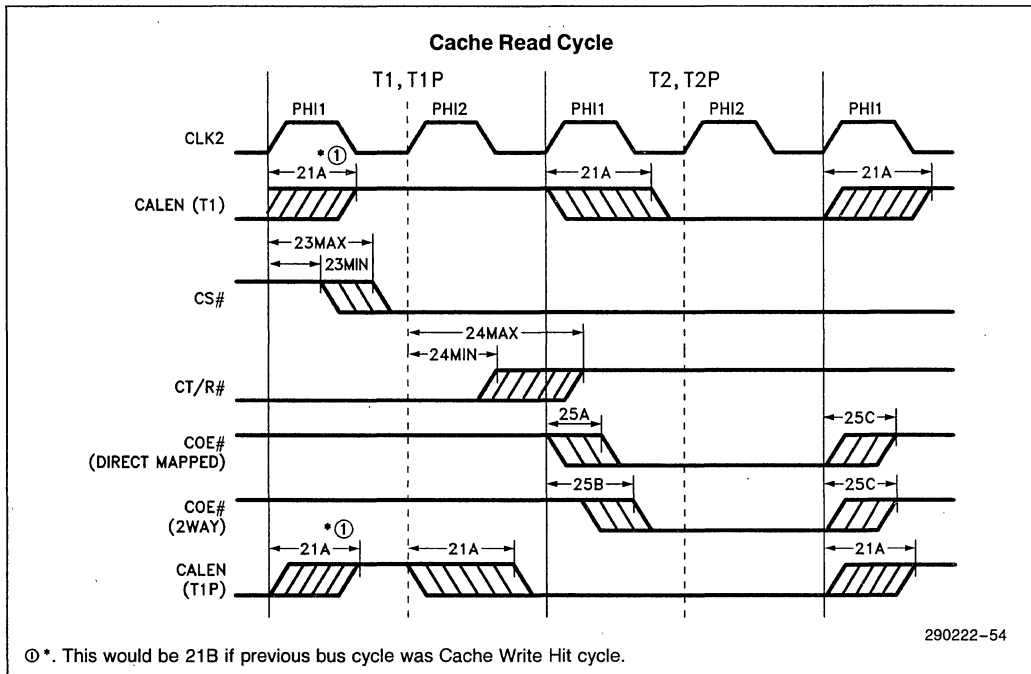
290222-52

Cache Read Miss (Cache Update Cycle)



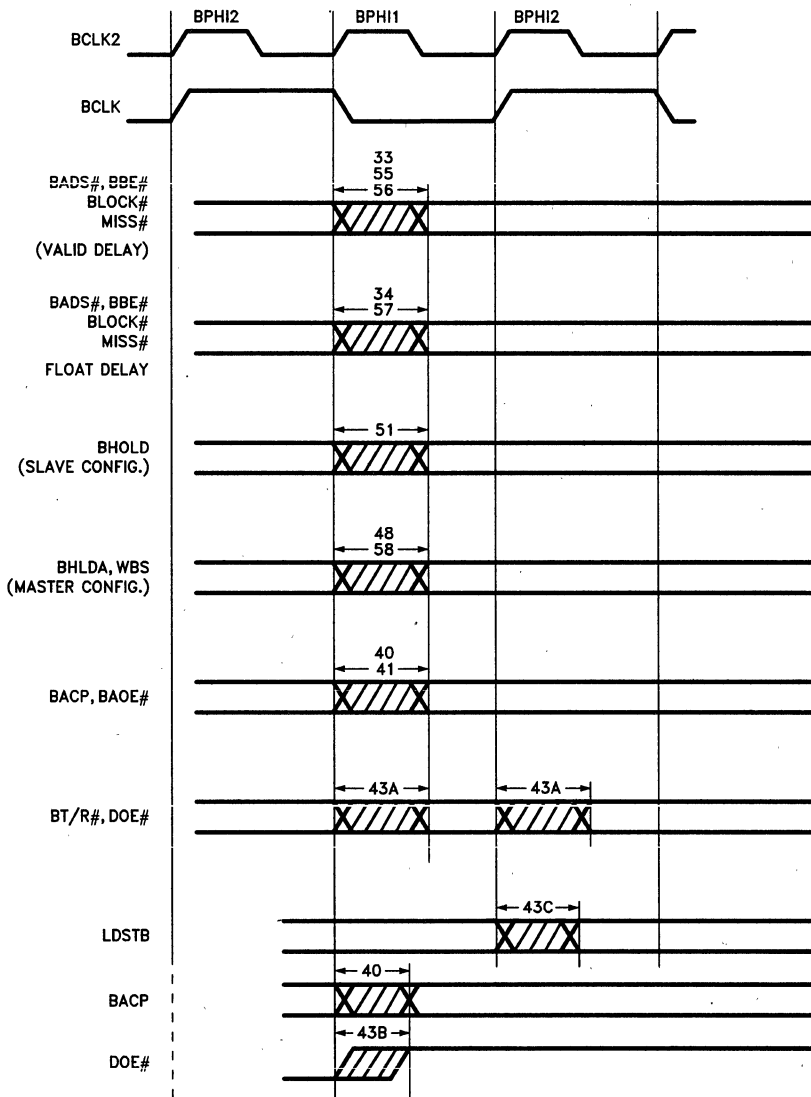
①*. This would be 21B if previous bus cycle was Cache Write Hit cycle.

290222-53



System Bus Interface Parameters (Continued)

OUTPUT DELAYS



290222-56

APPENDIX A

82385 SX Signal Summary

Signal Group/Name	Signal Function	Active State	Input/Output	Tri-State Output?
386 SX INTERFACE				
RESET	386 SX Reset	High	I	—
A1–A23	386 SX Address Bus	High	I	—
BHE #, BLE #	386 SX Byte Enables	Low	I	—
CLK2	386 SX Clock	—	I	—
READYO #	Ready Output	Low	O	No
BRDYEN #	Bus Ready Enable	Low	O	No
READYI #	386 SX Ready Input	Low	I	—
ADS #	386 SX Address Status	Low	I	—
M/IO #	386 SX Memory / I/O Indication	—	I	—
W/R #	386 SX Write/Read Indication	—	I	—
D/C #	386 SX Data/Control Indication	—	I	—
LOCK #	386 SX Lock Indication	Low	I	—
NA #	386 SX Next Address Request	Low	O	No
CACHE CONTROL				
CALEN	Cache Address Latch Enable	High	O	No
CT/R #	Cache Transmit/Receive	—	O	No
CS0 #, CS1 #	Cache Chip Selects	Low	O	No
COEA #, COEB #	Cache Output Enables	Low	O	No
CWEA #, CWEB #	Cache Write Enables	Low	O	No
LOCAL DECODE				
LBA #	386 SX Local Bus Access	Low	I	—
NCA #	Non-Cacheable Access	Low	I	—
STATUS AND CONTROL				
MISS #	Cache Miss Indication	Low	O	Yes
WBS	Write Buffer Status	High	O	No
FLUSH	Cache Flush	High	I	—

82385SX Signal Summary (Continued)

Signal Group/Name	Signal Function	Active State	Input/Output	Tri-State Output?
82385SX INTERFACE				
BREADY#	82385SX Ready Input	Low	I	—
BNA#	82385SX Next Address Request	Low	I	—
BLOCK#	82385SX Lock Indication	Low	O	Yes
BADS#	82385SX Address Status	Low	O	Yes
BBHE#, BBLE#	82385SX Byte Enables	Low	O	yes
DATA/ADDR CONTROL				
LDSTB	Local Data Strobe	Pos. Edge	O	No
DOE#	Data Output Enable	Low	O	No
BT/R#	Bus Transmit/Receive	—	O	No
BACP	Bus Address Clock Pulse	Pos. Edge	O	No
BAOE#	Bus Address Output Enable	Low	O	No
CONFIGURATION				
2W/D#	2-Way/Direct Map Select	—	I	—
M/S#	Master/Slave Select	—	I	—
DEFOE#	Define Cache Output Enable	—	I	—
COHERENCY				
SA1-SA23	Snoop Address Bus	High	I	—
SSTB#	Snoop Strobe	Low	I	—
SEN	Snoop Enable	High	I	—
ARBITRATION				
BHOLD	Hold	High	I/O	No
BHLDA	Hold Acknowledge	High	I/O	No

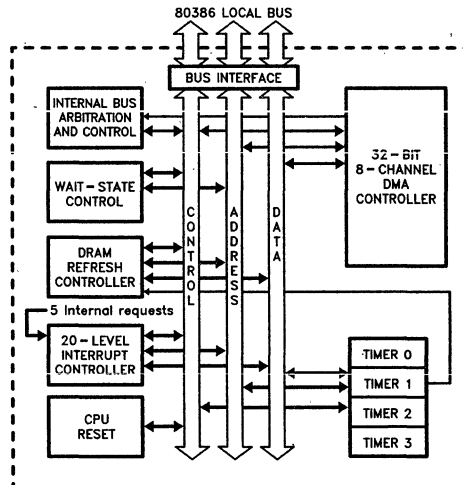


82380 HIGH PERFORMANCE 32-BIT DMA CONTROLLER WITH INTEGRATED SYSTEM SUPPORT PERIPHERALS

- High Performance 32-Bit DMA Controller
 - 50 MBytes/sec Maximum Data Transfer Rate at 25 MHz
 - 8 Independently Programmable Channels
- 20-Source Interrupt Controller
 - Individually Programmable Interrupt Vectors
 - 15 External, 5 Internal Interrupts
 - 82C59A Superset
- Four 16-Bit Programmable Interval Timers
 - 82C54 Compatible
- Programmable Wait State Generator
 - 0 to 15 Wait States Pipelined
 - 1 to 16 Wait States Non-Pipelined
- DRAM Refresh Controller
- 80386 Shutdown Detect and Reset Control!
 - Software/Hardware Reset
- High Speed CHMOS III Technology
- 132-Pin PGA Package
- Optimized for use with the 80386 Microprocessor
 - Resides on Local Bus for Maximum Bus Bandwidth

The 82380 is a multi-function support peripheral that integrates system functions necessary in an 80386 environment. It has eight channels of high performance 32-bit DMA with the most efficient transfer rates possible on the 80386 bus. System support peripherals integrated into the 82380 provide Interrupt Control, Timers, Wait State generation, DRAM Refresh Control, and System Reset logic.

The 82380's DMA Controller can transfer data between devices of different data path widths using a single channel. Each DMA channel operates independently in any of several modes. Each channel has a temporary data storage register for handling non-aligned data without the need for external alignment logic.



82380 Internal Block Diagram

290128-1

1.0 FUNCTIONAL OVERVIEW

The 82380 contains several independent functional modules. The following is a brief discussion of the components and features of the 82380. Each module has a corresponding detailed section later in this data sheet. Those sections should be referred to for design and programming information.

1.1 82380 Architecture

The 82380 is comprised of several computer system functions that are normally found in separate LSI and VLSI components. These include: a high-performance, eight-channel, 32-bit Direct Memory Access Controller; a 20-level Programmable Interrupt Controller which is a superset of the 82C59A; four 16-bit Programmable Interval Timers which are functionally equivalent to the 82C54 timers; a DRAM Refresh Controller; a Programmable Wait State Generator; and system reset logic. The interface to the 82380 is optimized for high-performance operation with the 80386 microprocessor.

The 82380 operates directly on the 80386 bus. In the Slave mode, it monitors the state of the proces-

sor at all times and acts or idles according to the commands of the host. It monitors the address pipeline status and generates the programmed number of wait states for the device being accessed. The 82380 also has logic to reset the 80386 via hardware or software reset requests and processor shutdown status.

After a system reset, the 82380 is in the Slave mode. It appears to the system as an I/O device. It becomes a bus master when it is performing DMA transfers.

To maintain compatibility with existing software, the registers within the 82380 are accessed as bytes. If the internal logic of the 82380 requires a delay before another access by the processor, wait states are automatically inserted into the access cycle. This allows the programmer to write initialization routines, etc. without regard to hardware recovery times.

Figure 1-1 shows the basic architectural components of the 82380. The following sections briefly discuss the architecture and function of each of the distinct sections of the 82380.

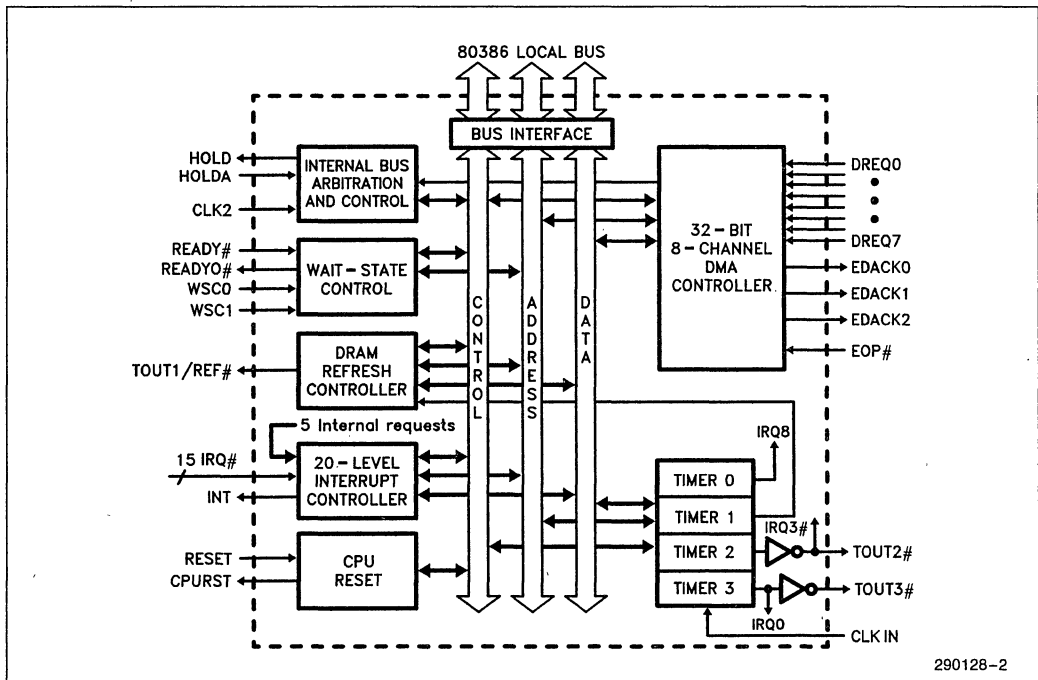


Figure 1-1. Architecture of the 82380

1.1.1 DMA CONTROLLER

The 82380 contains a high-performance, 8-channel, 32-bit DMA controller. It is capable of transferring any combination of bytes, words, and double words. The addresses of both source and destination can be independently incremented, decremented or held constant, and cover the entire 32-bit physical address space of the 80386. It can disassemble and assemble misaligned data via a 32-bit internal temporary data storage register. Data transferred between devices of different data path widths can also be assembled and disassembled using the internal temporary data storage register. The DMA Controller can also transfer aligned data between I/O and memory on the fly, allowing data transfer rates up to 32 megabytes per second for an 82380 operating at 16 MHz. Figure 1-2 illustrates the functional components of the DMA Controller.

There are twenty-four general status and command registers in the 82380 DMA Controller. Through these registers any of the channels may be programmed into any of the possible modes. The operating modes of any one channel are independent of the operation of the other channels.

Each channel has three programmable registers which determine the location and amount of data to be transferred:

Byte Count Register—Number of bytes to transfer. (24-bits)

Requester Register—Address of memory or peripheral which is requesting DMA service. (32-bits)

Target Register—Address of peripheral or memory which will be accessed. (32-bits)

There are also port addresses which, when accessed, cause the 82380 to perform specific functions. The actual data written does not matter, the act of writing to the specific address causes the command to be executed. The commands which operate in this mode are: Master Clear, Clear Terminal Count Interrupt Request, Clear Mask Register, and Clear Byte Pointer Flip-Flop.

DMA transfers can be done between all combinations of memory and I/O; memory-to-memory, memory-to-I/O, I/O-to-memory, and I/O-to-I/O. DMA service can be requested through software and/or hardware. Hardware DMA acknowledge signals are available for all channels (except channel 4) through an encoded 3-bit DMA acknowledge bus (EDACK0-2).

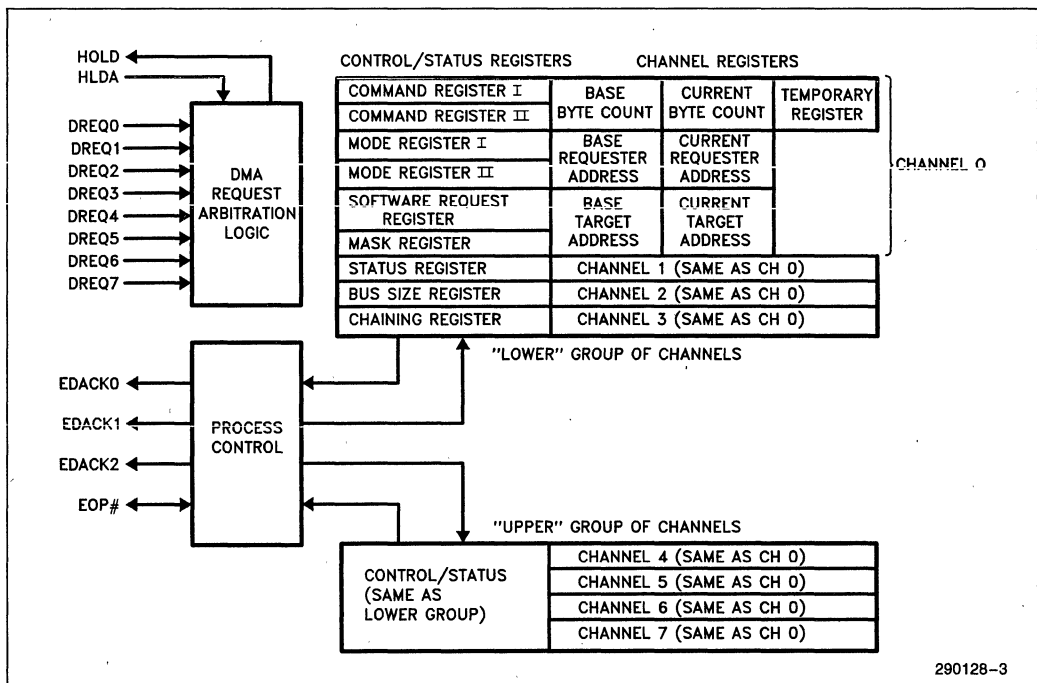


Figure 1-2. 82380 DMA Controller

The 82380 DMA controller transfers blocks of data (buffers) in three modes: Single Buffer, Buffer Auto-Initialize, and Buffer Chaining. In the Single Buffer Process, the 82380 DMA Controller is programmed to transfer one particular block of data. Successive transfers then require reprogramming of the DMA channel. Single Buffer transfers are useful in systems where it is known at the time the transfer begins what quantity of data is to be transferred, and there is a contiguous block of data area available.

The Buffer Auto-Initialize Process allows the same data area to be used for successive DMA transfers without having to reprogram the channel.

The Buffer Chaining Process allows a program to specify a list of buffer transfers to be executed. The 82380 DMA Controller, through interrupt routines, is reprogrammed from the list. The channel is reprogrammed for a new buffer before the current buffer transfer is complete. This pipelining of the channel programming process allows the system to allocate non-contiguous blocks of data storage space, and transfer all of the data with one DMA process. The buffers that make up the chain do not have to be in contiguous locations.

Channel priority can be fixed or rotating. Fixed priority allows the programmer to define the priority of DMA channels based on hardware or other fixed parameters. Rotating priority is used to provide peripherals access to the bus on a shared basis.

With fixed priority, the programmer can set any channel to have the current lowest priority. This al-

lows the user to reset or manually rotate the priority schedule without reprogramming the command registers.

1.1.2 PROGRAMMABLE INTERVAL TIMERS

Four 16-bit programmable interval timers reside within the 82380. These timers are identical in function to the timers in the 82C54 Programmable Interval Timer. All four of the timers share a common clock input which can be independent of the system clock. The timers are capable of operating in six different modes. In all of the modes, the current count can be latched and read by the 80386 at any time, making these very versatile event timers. Figure 1-3 shows the functional components of the Programmable Interval Timers.

The outputs of the timers are directed to key system functions, making system design simpler. Timer 0 is routed directly to an interrupt input and is not available externally. This timer would typically be used to generate time-keeping interrupts.

Timers 1 and 2 have outputs which are available for general timer/counter purposes as well as special functions. Timer 1 is routed to the refresh control logic to provide refresh timing. Timer 2 is connected to an interrupt request input to provide other timer functions. Timer 3 is a general purpose timer/counter whose output is available to external hardware. It is also connected internally to the interrupt request which defaults to the highest priority (IRQ0).

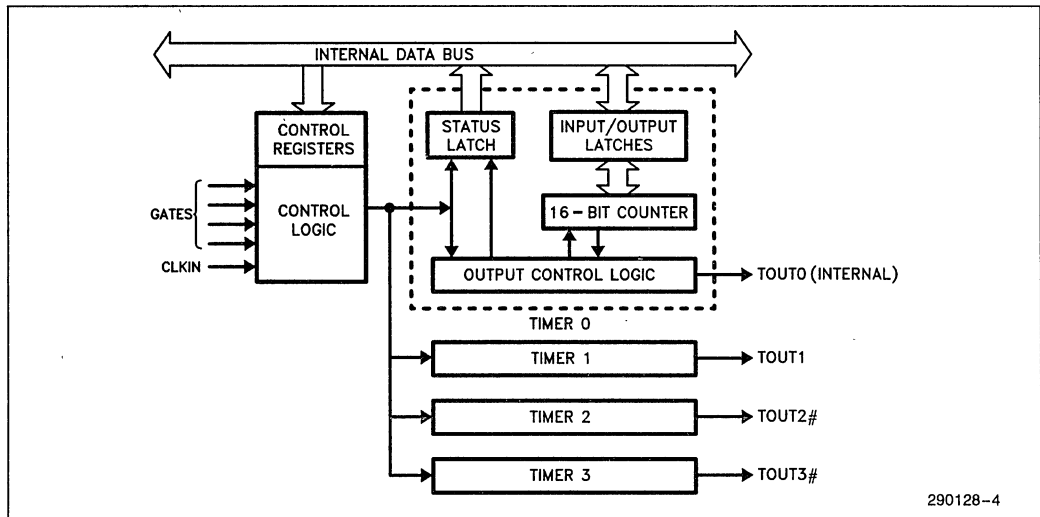


Figure 1-3. Programmable Interval Timers—Block Diagram

1.1.3 INTERRUPT CONTROLLER

The 82380 has the equivalent of three enhanced 82C59A Programmable Interrupt Controllers. These controllers can all be operated in the Master mode, but the priority is always as if they were cascaded. There are 15 interrupt request inputs provided for the user, all of which can be inputs from external slave interrupt controllers. Cascading 82C59As to these request inputs allows a possible total of 120 external interrupt requests. Figure 1-4 is a block diagram of the 82380 Interrupt Controller.

Each of the interrupt request inputs can be individually programmed with its own interrupt vector, allowing more flexibility in interrupt vector mapping than was available with the 82C59A. An interrupt is provided to alert the system that an attempt is being

made to program the vectors in the method of the 82C59A. This provides compatibility of existing software that used the 82C59A or 8259A with new designs using the 82380.

In the event of an unrequested or otherwise erroneous interrupt acknowledge cycle, the 82380 Interrupt Controller issues a default vector. This vector, programmed by the system software, will alert the system of unsolicited interrupts of the 80386.

The functions of the 82380 Interrupt Controller are identical to the 82C59A, except in regards to programming the interrupt vectors as mentioned above. Interrupt request inputs are programmable as either edge or level triggered and are software maskable. Priority can be either fixed or rotating and interrupt requests can be nested.

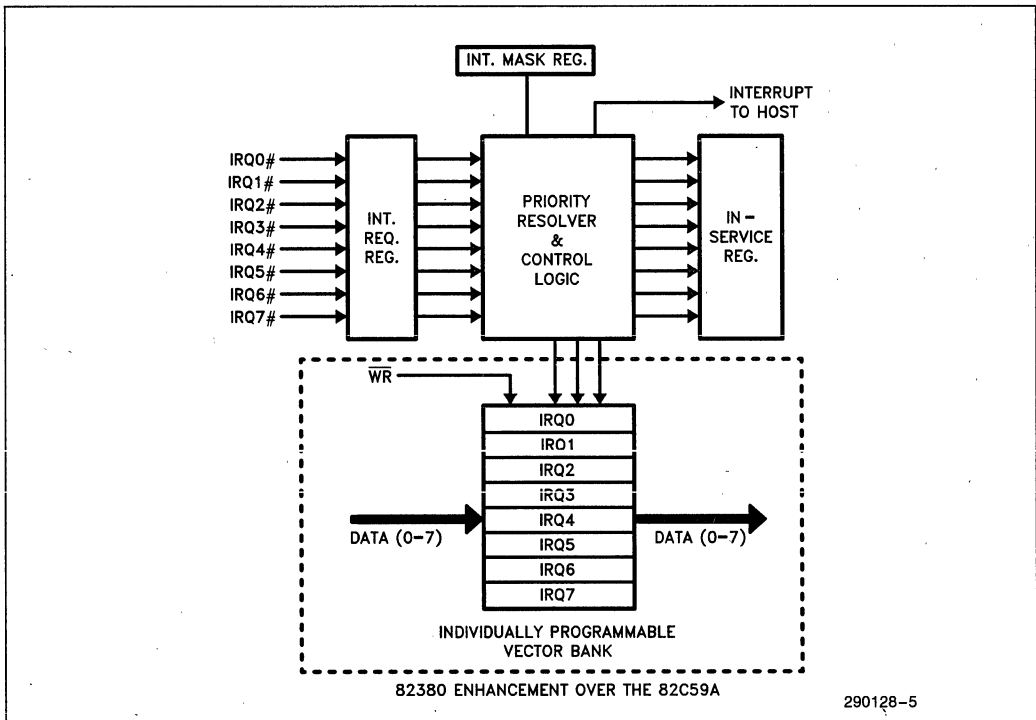


Figure 1-4. 82380 Interrupt Controller—Block Diagram

Enhancements are added to the 82380 for cascading external interrupt controllers. Master to Slave handshaking takes place on the data bus, instead of dedicated cascade lines.

1.1.4 WAIT STATE GENERATOR

The Wait State Generator is a programmable READY generation circuit for the 80386 bus. A peripheral requiring wait states can request the Wait State Generator to hold the processor's READY input inactive for a predetermined number of bus states. Six different wait state counts can be programmed into the Wait State Generator by software; three for memory accesses and three for I/O accesses. A block diagram of the 82380 Wait State Generator is shown in Figure 1-5.

The peripheral being accessed selects the required wait state count by placing a code on a 2-bit wait state select bus. This code along with the M/IO# signal from the bus master is used to select one of six internal 4-bit wait state registers which has been programmed with the desired number of wait states. From zero to fifteen wait states can be programmed into the wait state registers. The Wait State Generator tracks the state of the processor or current bus master at all times, regardless of which device is the current bus master and regardless of whether or not the Wait State Generator is currently active.

The 82380 Wait State Generator is disabled by making the select inputs both high. This allows hardware which is intelligent enough to generate its own ready signal to be accessed without penalty. As previously

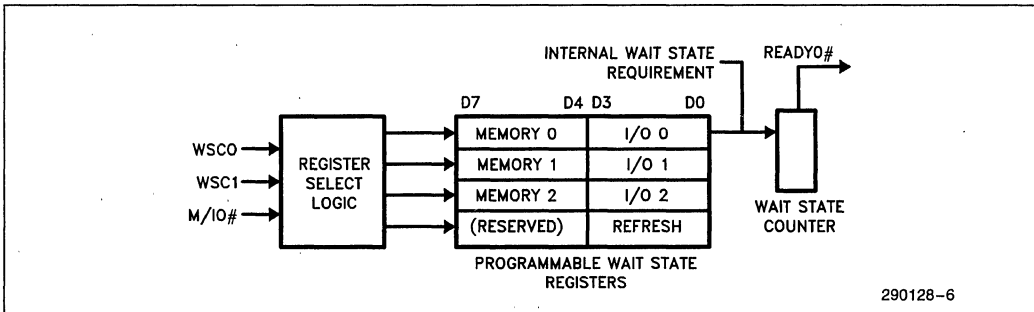
mentioned, deselecting the Wait State Generator does not disable its ability to determine the proper number of wait states due to pipeline status in subsequent bus cycles.

The number of wait states inserted into a pipelined bus cycle is the value in the selected wait state register. If the bus master is operating in the non-pipelined mode, the Wait State Generator will increase the number of wait states inserted into the bus cycle by one.

On reset, the Wait State Generator's registers are loaded with the value FFH, giving the maximum number of wait states for any access in which the wait state select inputs are active.

1.1.5 DRAM REFRESH CONTROLLER

The 82380 DRAM Refresh Controller consists of a 24-bit refresh address counter and bus arbitration logic. The output of Timer 1 is used to periodically request a refresh cycle. When the controller receives the request, it requests access to the system bus through the HOLD signal. When bus control is acknowledged by the processor or current bus master, the refresh controller executes a memory read operation at the address currently in the Refresh Address Register. At the same time, it activates a refresh signal (REF#) that the memory uses to force a refresh instead of a normal read. Control of the bus is transferred to the processor at the completion of this cycle. Typically a refresh cycle will take six clock cycles to execute on an 80386 bus.



290128-6

Figure 1-5. 82380 Wait State Generator—Block Diagram

The 82380 DRAM Refresh Controller has the highest priority when requesting bus access and will interrupt any active DMA process. This allows large blocks of data to be moved by the DMA controller without affecting the refresh function. Also the DMA controller is not required to completely relinquish the bus, the refresh controller simply steals a bus cycle between DMA accesses.

The amount by which the refresh address is incremented is programmable to allow for different bus widths and memory bank arrangements.

1.1.6 CPU RESET FUNCTION

The 82380 contains a special reset function which can respond to hardware reset signals from the 82384, as well as a software reset command. The circuit will hold the 80386's RESET line active while an external hardware reset signal is present at its RESET input. It can also reset the 80386 processor as the result of a software command. The software reset command causes the 82380 to hold the processor's RESET line active for a minimum of 62 CLK2 cycles; enough time to allow an 80386 to re-initialize.

The 82380 can be programmed to sense the shutdown detect code on the status lines from the 80386. If the Shutdown Detect function is enabled, the 82380 will automatically reset the processor. A diagnostic register is available which can be used to determine the cause of reset.

1.1.7 REGISTER MAP RELOCATION

After a hardware reset, the internal registers of the 82380 are located in I/O space beginning at port address 0000H. The map of the 82380's registers is relocatable via a software command. The default mapping places the 82380 between I/O addresses 0000H and 00DBH. The relocation register allows this map to be moved to any even 256-byte boundary in the processor's 16-bit I/O address space or any even 16-Mbyte boundary in the 32-bit memory address space.

1.2 Host Interface

The 82380 is designed to operate efficiently on the local bus of an 80386 microprocessor. The control

signals of the 82380 are identical in function to those of the 80386. As a slave, the 82380 operates with all of the features available on the 80386 bus. When the 82380 is in the Master mode, it looks identical to the 80386 to the connected devices.

The 82380 monitors the bus at all times, and determines whether the current bus cycle is a pipelined or non-pipelined access. All of the status signals of the processor are monitored.

The control, status, and data registers within the 82380 are located at fixed addresses relative to each other, but the group can be relocated to either memory or I/O space and to different locations within those spaces.

As a Slave device, the 82380 monitors the control/status lines of the CPU. The 82380 will generate all of the wait states it needs whenever it is accessed. This allows the programmer the freedom of accessing 82380 registers without having to insert NOPs in the program to wait for slower 82380 internal registers.

The 82380 can determine if a current bus cycle is a pipelined or a non-pipelined cycle. It does this by monitoring the ADS# and READY# signals and thereby keeping track of the current state of the 80386.

As a bus master, the 82380 looks like an 80386 to the rest of the system. This enables the designer greater flexibility in systems which include the 82380. The designer does not have to alter the interfaces of any peripherals designed to operate with the 80386 to accommodate the 82380. The 82380 will access any peripherals on the bus in the same manner as the 80386, including recognizing pipelined bus cycles.

The 82380 is accessed as an 8-bit peripheral. This is done to maintain compatibility with existing system architectures and software. The 80386 places the data of all 8-bit accesses either on D (0-7) or D (8-15). The 82380 will only accept data on these lines when in the Slave mode. When in the Master mode, the 82380 is a full 32-bit machine, sending and receiving data in the same manner as the 80386.

1.3 IBM PC* System Compatibility

The 82380 is an 80386 companion device designed to provide an enhancement of the system functions common to most small computer systems. It is modeled after and is a superset of the Intel peripheral products found in the IBM PC, PC-AT, and other popular small computers.

2.0 80386 HOST INTERFACE

The 82380 contains a set of interface signals to operate efficiently with the 80386 host processor. These signals were designed so that minimal hardware is needed to connect the 82380 to the 80386.

Figure 2-1 depicts a typical system configuration with the 80386 processor. As shown in the diagram, the 82380 is designed to interface directly with the 80386 bus.

*IBM PC and IBM PC-AT are registered trademarks of International Business Machines Inc.

Since the 82380 is residing on the opposite side of the data bus transceiver (with respect to the rest of the peripherals in the system), it is important to note that the transceiver should be controlled so that contention between the data bus transceiver and the 82380 will not occur. In order to do this, port address decoding logic should be included in the direction and enable control logic of the transceiver. When any of the 82380 internal registers is read, the data bus transceiver should be disabled so that only the 82380 will drive the local bus.

This section describes the basic bus functions of the 82380 to show how this device interacts with the 80386 processor. Other signals which are not directly related to the host interface will be discussed in their associated functional block description.

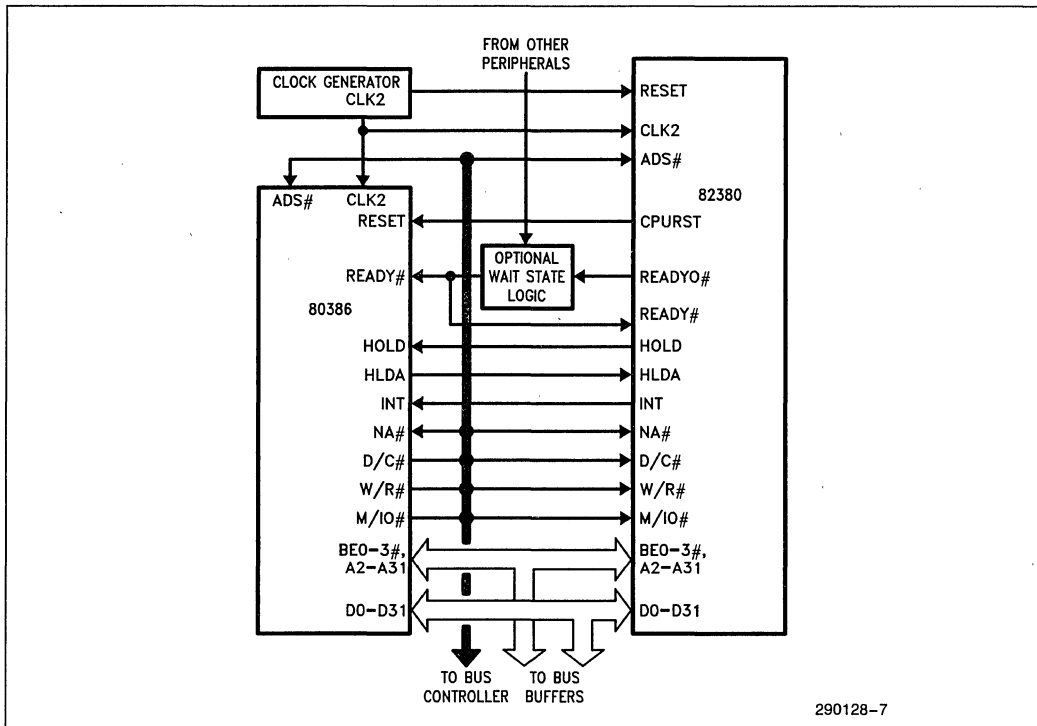


Figure 2-1. 80386/82380 System Configuration

2.1 Master and Slave Modes

At any time, the 82380 acts as either a Slave device or a Master device in the system. Upon reset, the 82380 will be in the Slave Mode. In this mode, the 80386 processor can read/write into the 82380 internal registers. Initialization information may be programmed into the 82380 during Slave Mode.

When DMA service (including DRAM Refresh Cycles generated by the 82380) is requested, the 82380 will request and subsequently get control of the 80386 local bus. This is done through the HOLD and HLDA (Hold Acknowledge) signals. When the 80386 processor responds by asserting the HLDA signal, the 82380 will switch into Master Mode and perform DMA transfers. In this mode, the 82380 is the bus master of the system. It can read/write data from/to memory and peripheral devices. The 82380 will return to the Slave Mode upon completion of DMA transfers, or when HLDA is negated.

2.2 80386 Interface Signals

As mentioned in the Architecture section, the Bus Interface module of the 82380 (see Figure 1-1) contains signals that are directly connected to the 80386 host processor. This module has separate 32-bit Data and Address busses. Also, it has additional control signals to support different bus operations on the system. By residing on the 80386 local bus, the 82380 shares the same address, data and control lines with the processor. The following subsections discuss the signals which interface to the 80386 host processor.

2.2.1 CLOCK (CLK2)

The CLK2 input provides fundamental timing for the 82380. It is divided by two internally to generate the 82380 internal clock. Therefore, CLK2 should be driven with twice the 80386's frequency. In order to maintain synchronization with the 80386 host processor, the 82380 and the 80386 should share a common clock source.

The internal clock consists of two phases: PHI1 and PHI2. Each CLK2 period is a phase of the internal clock. PHI2 is usually used to sample input and set up internal signals and PHI1 is for latching internal data. Figure 2-2 illustrates the relationship of CLK2 and the 82380 internal clock signals. The CPURST signal generated by the 82380 guarantees that the 80386 will wake up in phase with PHI1.

2.2.2 DATA BUS (D0-D31)

This 32-bit three-state bidirectional bus provides a general purpose data path between the 82380 and the system. These pins are tied directly to the corresponding Data Bus pins of the 80386 local bus. The Data Bus is also used for interrupt vectors generated by the 82380 in the Interrupt Acknowledge cycle.

During Slave I/O operations, the 82380 expects a single byte to be written or read. When the 80386 host processor writes into the 82380, either D0-D7 or D8-D15 will be latched into the 82380, depending upon how the Byte Enable (BE0#-BE#3) signals are driven. The 82380 does not need to look at D16-D31 since the 80386 duplicates the single byte

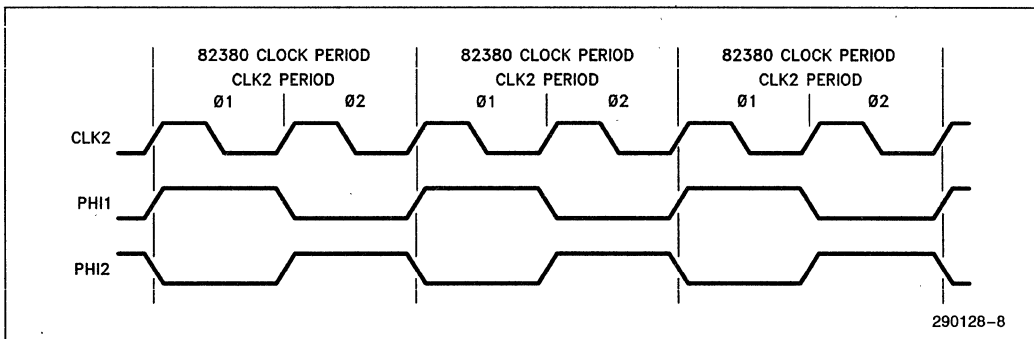


Figure 2-2. CLK2 and 82380 Internal Clock

data on both halves of the bus. When the 80386 host processor reads from the 82380, the single byte data will be duplicated four times on the Data Bus; i.e., on D0-D7, D8-D15, D16-D23 and D24-D31.

During Master Mode, the 82380 can transfer 32-, 16-, and 8-bit data between memory (or I/O devices) and I/O devices (or memory) via the Data Bus.

2.2.3 ADDRESS BUS (A31-A2)

These three-state bidirectional signals are connected directly to the 80386 Address Bus. In the Slave Mode, they are used as input signals so that the processor can address the 82380 internal ports/registers. In the Master Mode, they are used as output signals by the 82380 to address memory and peripheral devices. The Address Bus is capable of addressing 4 G-bytes of physical memory space (00000000H to FFFFFFFFH), and 64 K-bytes of I/O addresses (00000000H to 0000FFFFH).

2.2.4 BYTE ENABLE (BE3#-BE0#)

These bidirectional pins select specific byte(s) in the double word addressed by A31-A2. Similar to the Address Bus function, these signals are used as inputs to address internal 82380 registers during Slave Mode operation. During Master Mode operation, they are used as outputs by the 82380 to address memory and I/O locations.

NOTE:

In addition to the above function, BE3# is used to enable a production test mode and must be LOW during reset. The 80386 processor will automatically hold BE3# LOW during RESET.

The definitions of the Byte Enable signals depend upon whether the 82380 is in the Master or Slave Mode. These definitions are depicted in Table 2-1.

Table 2-1. Byte Enable Signals

As INPUTS (Slave Mode):

BE3#-BE0#	Implied A1, A0	Data Bits Written to 82380*
XXX0	00	D0-D7
XX01	01	D8-D15
X011	10	D0-D7
X111	11	D8-D15

X-DON'T CARE

*During READ, data will be duplicated on D0-D7, D8-D15, D16-D23, and D24-D31.

During WRITE, the 80386 host processor duplicates data on D0-D15, and D16-D31, so that the 82380 is concerned only with the lower half of the Data Bus.

As OUTPUTS (Master Mode):

BE3#-BE0#	Byte to be Accessed Relative to A31-A2	Logical Byte Presented On Data Bus During WRITE Only*			
		D24-31	D16-23	D8-15	D0-7
1110	0	U	U	U	A
1101	1	U	U	A	A
1011	2	U	A	U	A
0111	3	A	U	A	A
1001	1, 2	U	B	A	A
1100	0, 1	U	U	B	A
0011	2, 3	B	A	B	A
1000	0, 1, 2	U	C	B	A
0001	1, 2, 3	C	B	A	A
0000	0, 1, 2, 3	D	C	B	A

U = Undefined

A = Logical D0-D7

B = Logical D8-D15

C = Logical D16-D23

D = Logical D24-D31

*Actual number of bytes accessed depends upon the programmed data path width.

2.2.5 BUS CYCLE DEFINITION SIGNALS (D/C#, W/R#, M/IO#)

These three-state bidirectional signals define the type of bus cycle being performed. W/R# distinguishes between write and read cycles. D/C# distinguishes between processor data and control cycles. M/IO# distinguishes between memory and I/O cycles.

During Slave Mode, these signals are driven by the 80386 host processor; during Master Mode, they are driven by the 82380. In either mode, these signals will be valid when the Address Status (ADS#) is driven LOW. Exact bus cycle definitions are given in Table 2-2. Note that some combinations are recognized as inputs, but not generated as outputs. In the Master Mode, D/C# is always HIGH.

2.2.6 ADDRESS STATUS (ADS#)

This bidirectional signal indicates that a valid address (A2-A31, BE0#-BE3#) and bus cycle definition (W/R#, D/C#, M/IO#) is being driven on the bus. In the Master Mode, it is driven by the 82380 as an output. In the Slave Mode, this signal is monitored as an input by the 82380. By the current and past status of ADS# and the READY# input, the 82380 is able to determine, during Slave Mode, if the next bus cycle is a pipelined address cycle. ADS# is asserted during T1 and T2P bus states (see Bus State Definition).

Note that during the idle states at the beginning and the end of a DMA process, neither the 80386 nor the 82380 is driving the ADS# signal; i.e., the signal is left floated. Therefore, it is important to use a pull-up resistor (approximately 10 KΩ) on the ADS# signal.

2.2.7 TRANSFER ACKNOWLEDGE (READY#)

This input indicates that the current bus cycle is complete. In the Master Mode, assertion of this sig-

nal indicates the end of a DMA bus cycle. In the Slave Mode, the 82380 monitors this input and ADS# to detect a pipelined address cycles. This signal should be tied directly to the READY# input of the 80386 host processor.

2.2.8 NEXT ADDRESS REQUEST (NA#)

This input is used to indicate to the 82380 in the Master Mode that the system is requesting address pipelining. When driven LOW by either memory or peripheral devices during Master Mode, it indicates that the system is prepared to accept a new address and bus cycle definition signals from the 82380 before the end of the current bus cycle. If this input is active when sampled by the 82380, the next address is driven onto the bus, provided a bus request is already pending internally.

This input pin is monitored only in the Master Mode. In the Slave Mode, the 82380 uses the ADS# and READY# signals to determine address pipelining cycles, and NA# will be ignored.

2.2.9 RESET (RESET, CPURST)

RESET

This synchronous input suspends any operation in progress and places the 82380 in a known initial state. Upon reset, the 82380 will be in the Slave Mode waiting to be initialized by the 80386 host processor. The 82380 is reset by asserting RESET for 15 or more CLK2 periods. When RESET is asserted, all other input pins are ignored, and all other bus pins are driven to an idle bus state as shown in Table 2-3. The 82380 will determine the phase of its internal clock following RESET going inactive.

Table 2-2. Bus Cycle Definition

M/IO#	D/C#	W/R#	As INPUTS	As OUTPUTS
0	0	0	Interrupt Acknowledge	NOT GENERATED
0	0	1	UNDEFINED	NOT GENERATED
0	1	0	I/O Read	I/O Read
0	1	1	I/O Write	I/O Write
1	0	0	UNDEFINED	NOT GENERATED
1	0	1	HALT if BE(3-0) # = X011 SHUTDOWN if BE (3-0) # = XXX0	NOT GENERATED
1	1	0	Memory Read	Memory Read
1	1	1	Memory Write	Memory Write

Table 2-3. Output Signals Following RESET

Signal	Level
A2–A31, D0–D31, BE0# –BE3#	Float
D/C#, W/R#, M/IO#, ADS#	Float
READYO#	'1'
EOP#	'1' (Weak Pull-UP)
EDACK2–EDACK0	'100'
HOLD	'0'
INT	UNDEFINED*
TOUT1/REF#, TOUT2#/IRQ3#, TOUT3#	UNDEFINED*
CPURST	'0'

*The Interrupt Controller and Programmable Interval Timer are initialized by software commands.

RESET is level-sensitive and must be synchronous to the CLK2 signal. Therefore, this RESET input should be tied to the RESET output of the Clock Generator. The RESET setup and hold time requirements are shown in Figure 2.3.

CPURST

This output signal is used to reset the 80386 host processor. It will go active (HIGH) whenever one of the following events occurs: a) 82380's RESET input is active; b) a software RESET command is issued to the 82380; or c) when the 82380 detects a processor Shutdown cycle and when this detection feature is enabled (see CPU Reset and Shutdown Detect). When activated, CPURST will be held active for 62 CLK2 periods. The timing of CPURST is such that the 80386 processor will be in synchronization with the 82380. This timing is shown in Figure 2-4.

2.2.10 INTERRUPT OUT (INT)

This output pin is used to signal the 80386 host processor that one or more interrupt requests (either internal or external) are pending. The processor is expected to respond with an Interrupt Acknowledge cycle. This signal should be connected directly to the Maskable Interrupt Request (INTR) input of the 80386 host processor.

2.3 82380 Bus Timing

The 82380 internally divides the CLK2 signal by two to generate its internal clock. Figure 2-2 shows the relationship of CLK2 and the internal clock. The internal clock consists of two phases: PHI1 and PHI2. Each CLK2 period is a phase of the internal clock. In Figure 2-2, both PHI1 and PHI2 of the 82380 internal clock are shown.

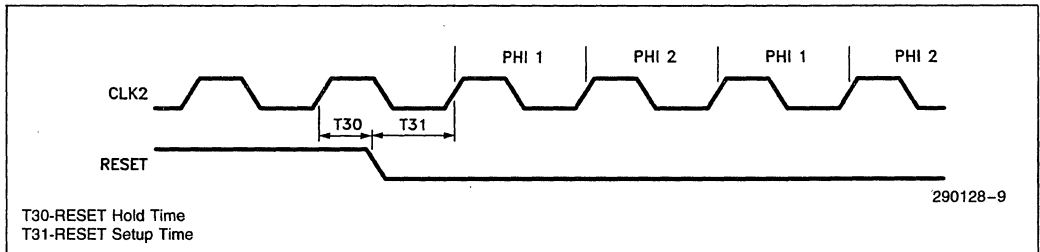


Figure 2-3. RESET Timing

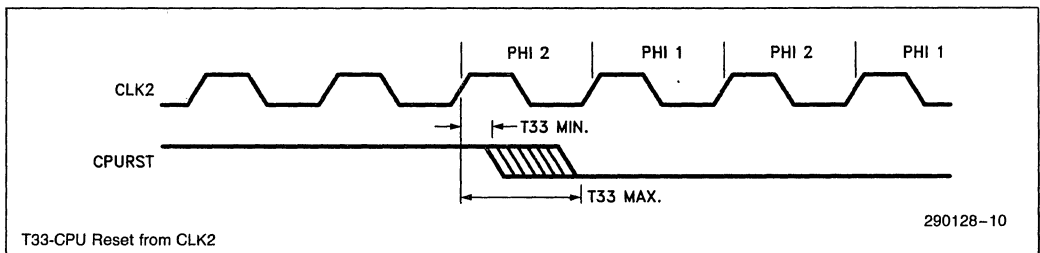


Figure 2-4. CPURST Timing

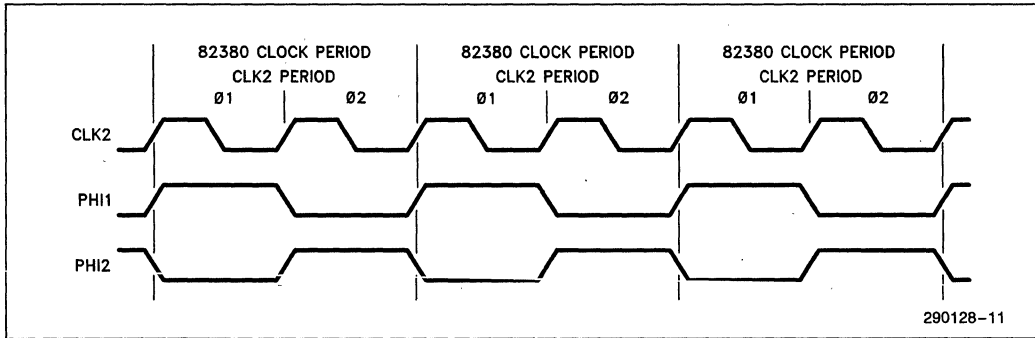


Figure 2-2. CLK2 and 82380 Internal Clock

In the 82380, whether it is in the Master or Slave Mode, the shortest time unit of bus activity is a bus state. A bus state, which is also referred to as a 'T-state', is defined as one 82380 PHI2 clock period (i.e., two CLK2 periods). Recall in Table 2-2, there are six different types of bus cycles in the 82380 as defined by the M/IO#, D/C# and W/R# signals. Each of these bus cycles is composed of two or more bus states. The length of a bus cycle depends on when the READY# input is asserted (i.e., driven LOW).

2.3.1 ADDRESS PIPELINING

The 82380 supports Address Pipelining as an option in both the Master and Slave Mode. This feature typically allows a memory or peripheral device to operate with one less wait state than would otherwise be required. This is possible because during a pipelined cycle, the address and bus cycle definition of the next cycle will be generated by the bus master while waiting for the end of the current cycle to be acknowledged. The pipelined bus is especially well suited for interleaved memory environment. For 16 MHz interleaved memory designs with 100 ns access time DRAMs, zero wait state memory accesses can be achieved when pipelined addressing is selected.

In the Master Mode, the 82380 is capable of initiating, on a cycle-by-cycle basis, either a pipelined or non-pipelined access depending upon the state of the NA# input. If a pipelined cycle is requested (indicated by NA# being driven LOW), the 82380 will

drive the address and bus cycle definition of the next cycle as soon as there is an internal bus request pending.

In the Slave Mode, the 82380 is constantly monitoring the ADS# and READY# signals on the processor local bus to determine if the current bus cycle is a pipelined cycle. If a pipelined cycle is detected, the 82380 will request one less wait state from the processor if the Wait State Generator feature is selected. On the other hand, during an 82380 internal register access in a pipelined cycle, it will make use of the advance address and bus cycle information. In all cases, Address Pipelining will result in a savings of one wait state.

2.3.2 MASTER MODE BUS TIMING

When the 82380 is in the Master Mode, it will be in one of six bus states. Figure 2-5 shows the complete bus state diagram of the Master Mode, including pipelined address states. As seen in the figure, the 82380 state diagram is very similar to that of the 80386. The major difference is that in the 82380, there is no Hold state. Also, in the 82380, the conditions for some state transitions depend upon whether it is the end of a DMA process*.

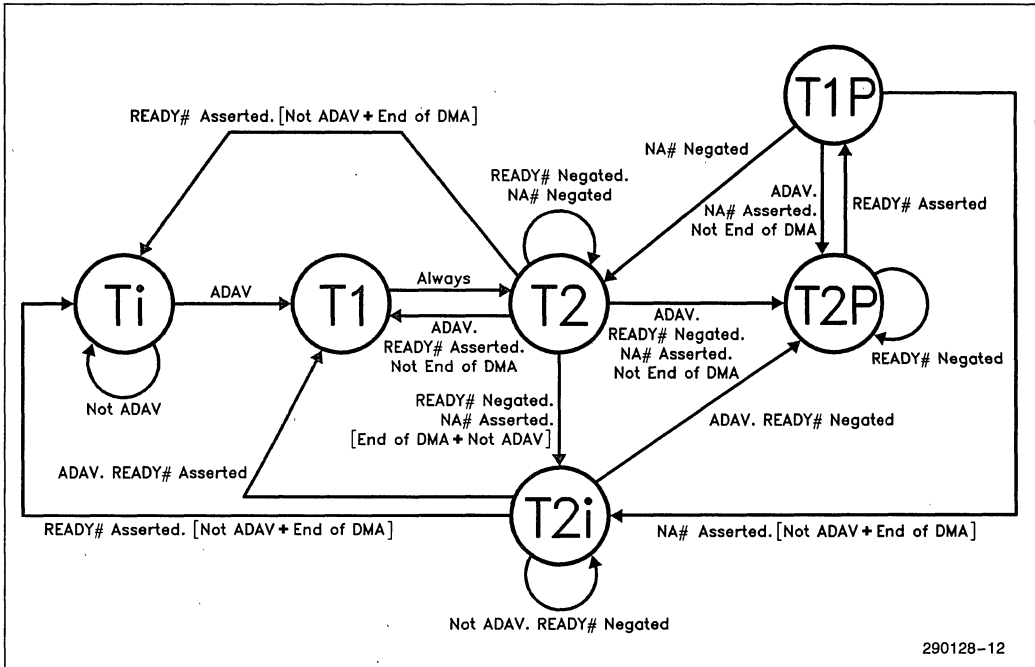
NOTE:

*The term 'end of a DMA process' is loosely defined here. It depends on the DMA modes of operation as well as the state of the EOP# and DREQ inputs. This is explained in detail in section 3—DMA Controller.

The 82380 will enter the idle state, Ti, upon RESET and whenever the internal address is not available at the end of a DMA cycle or at the end of a DMA process. When address pipelining is not used (NA# is not asserted), a new bus cycle always begins with state T1. During T1, address and bus cycle definition signals will be driven on the bus. T1 is always followed by T2.

If a bus cycle is not acknowledged (with READY#) during T2 and NA# is negated, T2 will be repeated. When the end of the bus cycle is acknowledged during T2, the following state will be T1 of the next bus cycle (if the internal address latch is loaded and if this is not the end of the DMA process). Otherwise, the Ti state will be entered. Therefore, if the memory or peripheral accessed is fast enough to respond within the first T2, the fastest non-pipelined cycle will take one T1 and one T2 state.

Use of the address pipelining feature allows the 82380 to enter three additional bus states: T1P, T2P, and T2i. T1P is the first bus state of a pipelined bus cycle. T2P follows T1P (or T2) if NA# is asserted when sampled. The 82380 will drive the bus with the address and bus cycle definition signals of the next cycle during T2P. From the state diagram, it can be seen that after an idle state Ti, the first bus cycle must begin with T1, and is therefore a non-pipelined bus cycle. The next bus cycle can be pipelined if NA# is asserted and the previous bus cycle ended in a T2P state. Once the 82380 is in a pipelined cycle and provided that NA# is asserted in subsequent cycles, the 82380 will be switching between T1P and T2P states. If the end of the current bus cycle is not acknowledged by the READY# input, the 82380 will extend the cycle by adding T2P states. The fastest pipelined cycle will consist of one T1P and one T2P state.



NOTE:
ADAV—Internal Address Available

Figure 2-5. Master Mode State Diagram

The 82380 will enter state T2i when NA# is asserted and when one of the following two conditions occurs. The first condition is when the 82380 is in state T2. T2i will be entered if READY# is not asserted and there is no next address available. This situation is similar to a wait state. The 82380 will stay in T2i for as long as this condition exists. The second condition which will cause the 82380 enter T2i is when the 82380 is in state T1P. Before going to

state T2P, the 82380 needs to wait in state T2i until the next address is available. Also, in both cases, if the DMA process is complete, the 82380 will enter the T2i state in order to finish the current DMA cycle.

Figure 2-6 is a timing diagram showing non-pipelined bus accesses in the Master Mode. Figure 2-7 shows the timing of pipelined accesses in the Master Mode.

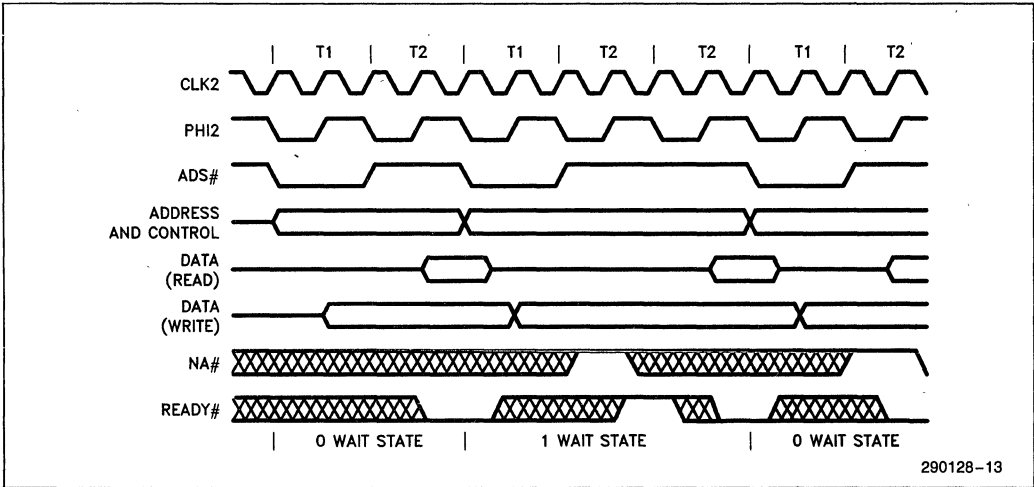


Figure 2-6. Non-Pipelined Bus Cycles

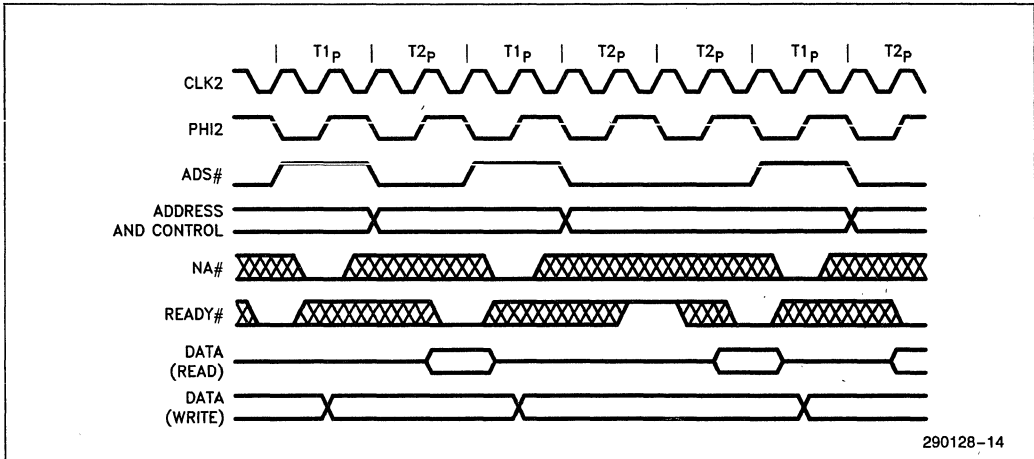


Figure 2-7. Pipelined Bus Cycles

2.3.3 SLAVE MODE BUS TIMING

Figure 2-8 shows the Slave Mode bus timing in both pipelined and non-pipelined cycles when the 82380 is being accessed. Recall that during Slave Mode, the 82380 will constantly monitor the ADS# and READY# signals to determine if the next cycle is pipelined. In Figure 2-8, the first cycle is non-pipelined and the second cycle is pipelined. In the pipelined cycle, the 82380 will start decoding the ad-

dress and bus cycle signals one bus state earlier than in a non-pipelined cycle.

The READY# input signal is sampled by the 80386 host processor to determine the completion of a bus cycle. This occurs during the end of every T2 and T2P state. Normally, the output of the 82380 Wait State Generator, READYO#, is directly connected to the READY# input of the 80386 host processor and the 82380. In such case, READYO# and READY# will be identical (see Wait State Generator).

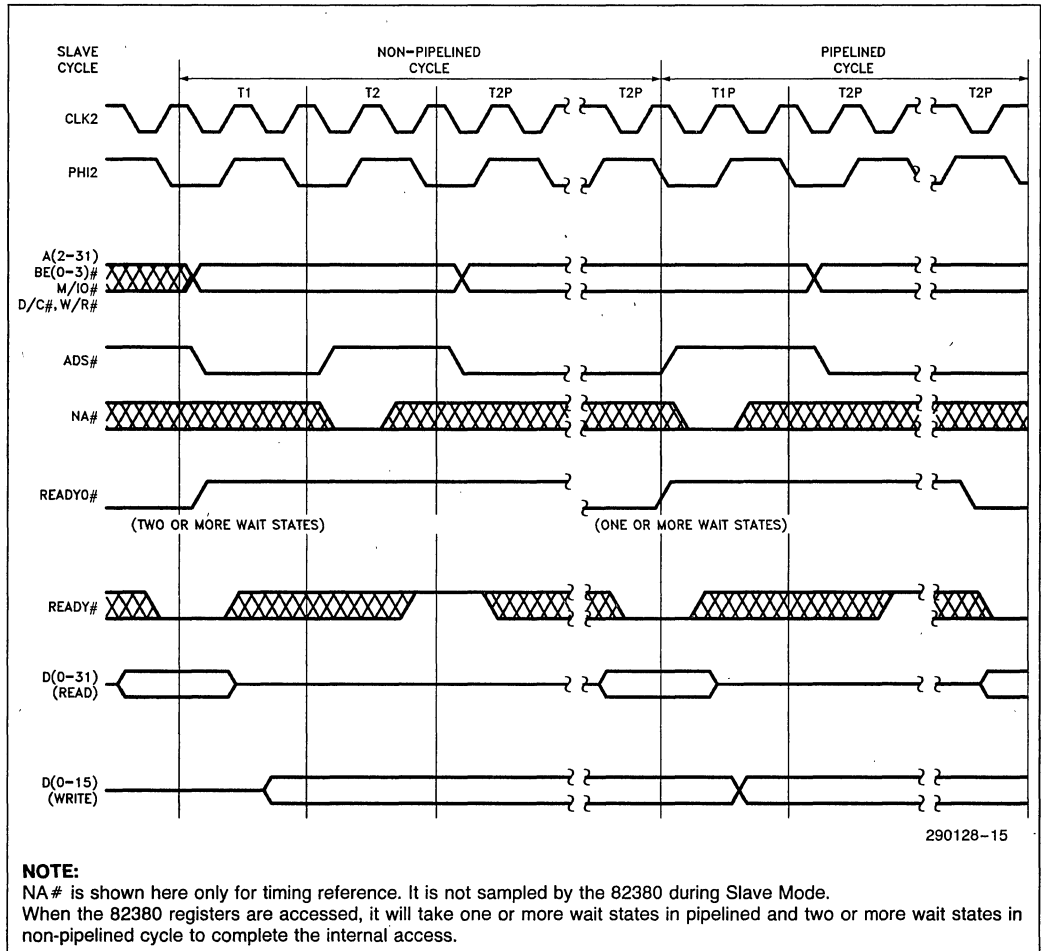


Figure 2-8. Slave Read/Write Timing

3.0 DMA Controller

The 82380 DMA Controller is capable of transferring data between any combination of memory and/or I/O, with any combination (8-, 16-, or 32-bits) of data path widths. Bus bandwidth is optimized through the use of an internal temporary register which can disassemble or assemble data to or from either an aligned or a non-aligned destination or source. Figure 3-1 is a block diagram of the 82380 DMA Controller.

The 82380 has eight channels of DMA. Each channel operates independently of the others. Within the operation of the individual channels, there are many different modes of data transfer available. Many of the operating modes can be intermixed to provide a very versatile DMA controller.

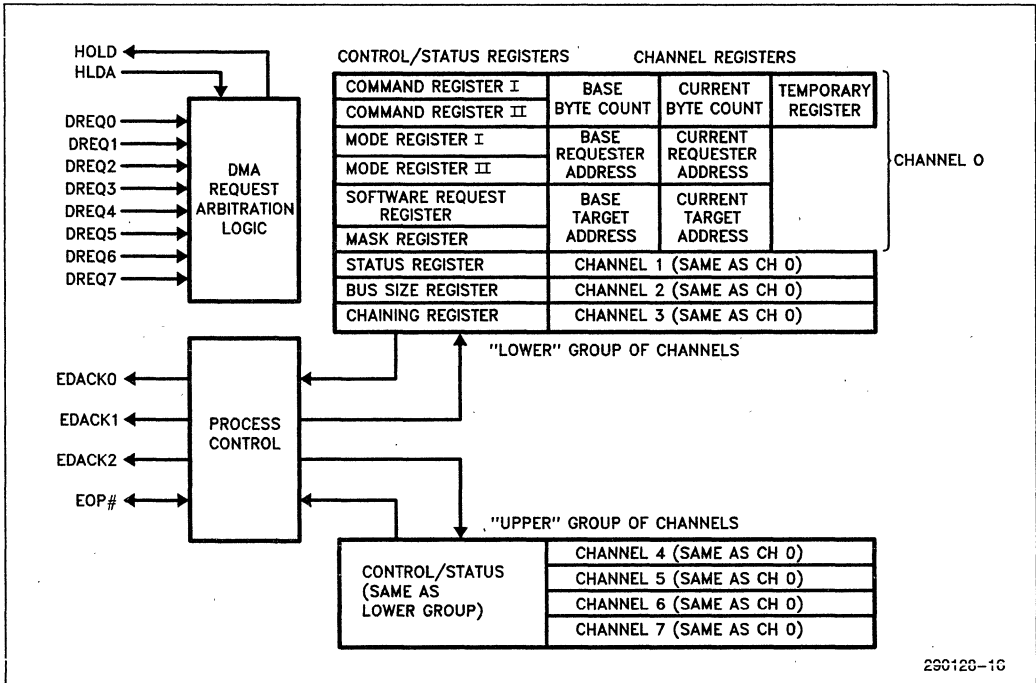


Figure 3-1. 82380 DMA Controller Block Diagram

3.1 Functional Description

In describing the operation of the 82380's DMA Controller, close attention to terminology is required. Before entering the discussion of the function of the 82380 DMA Controller, the following explanations of some of the terminology used herein may be of benefit. First, a few terms for clarification:

DMA PROCESS—A DMA process is the execution of a programmed DMA task from beginning to end. Each DMA process requires initial programming by the host 80386 microprocessor.

BUFFER—A contiguous block of data.

BUFFER TRANSFER—The action required by the DMA to transfer an entire buffer.

DATA TRANSFER—The DMA action in which a group of bytes, words, or double words are moved between devices by the DMA Controller. A data transfer operation may involve movement of one or many bytes.

BUS CYCLE—Access by the DMA to a single byte, word, or double word.

Each DMA channel consists of three major components. These components are identified by the contents of programmable registers which define the memory or I/O devices being serviced by the DMA. They are the Target, the Requester, and the Byte Count. They will be defined generically here and in greater detail in the DMA register definition section.

The Requester is the device which requires service by the 82380 DMA Controller, and makes the request for service. All of the control signals which the DMA monitors or generates for specific channels are logically related to the Requester. Only the Requester is considered capable of initiating or terminating a DMA process.

The Target is the device with which the Requester wishes to communicate. As far as the DMA process is concerned, the Target is a slave which is incapable of control over the process.

The direction of data transfer can be either from Requester to Target or from Target to Requester; i.e., each can be either a source or a destination.

The Requester and Target may each be either I/O or memory. Each has an address associated with it that can be incremented, decremented, or held constant. The addresses are stored in the Requester Address Registers and Target Address Registers,

respectively. These registers have two parts: one which contains the current address being used in the DMA process (Current Address Register), and one which holds the programmed base address (Base Address Register). The contents of the Base Registers are never changed by the 82380 DMA Controller. The Current Registers are incremented or decremented according to the progress of the DMA process.

The Byte Count is the component of the DMA process which dictates the amount of data which must be transferred. Current and Base Byte Count Registers are provided. The Current Byte Count Register is decremented once for each byte transferred by the DMA process. When the register is decremented past zero, the Byte Count is considered 'expired' and the process is terminated or restarted, depending on the mode of operation of the channel. The point at which the Byte Count expires is called 'Terminal Count' and several status signals are dependent on this event.

Each channel of the 82380 DMA Controller also contains a 32-bit Temporary Register for use in assembling and disassembling non-aligned data. The operation of this register is transparent to the user, although the contents of it may affect the timing of some DMA handshake sequences. Since there is data storage available for each channel, the DMA Controller can be interrupted without loss of data.

The 82380 DMA Controller is a slave on the bus until a request for DMA service is received via either a software request command or a hardware request signal. The host processor may access any of the control/status or channel registers at any time the 82380 is a bus slave. Figure 3-2 shows the flow of operations that the DMA Controller performs.

At the time a DMA service request is received, the DMA Controller issues a bus hold request to the host processor. The 82380 becomes the bus master when the host relinquishes the bus by asserting a hold acknowledge signal. The channel to be serviced will be the one with the highest priority at the time the DMA Controller becomes the bus master. The DMA Controller will remain in control of the bus until the hold acknowledge signal is removed, or until the current DMA transfer is complete.

While the 82380 DMA Controller has control of the bus, it will perform the required data transfer(s). The type of transfer, source and destination addresses, and amount of data to transfer are programmed in the control registers of the DMA channel which received the request for service.

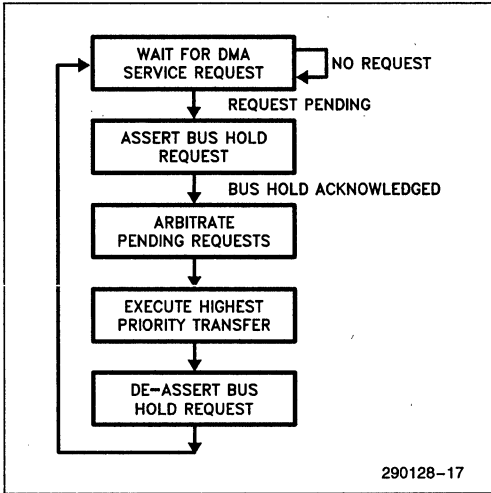


Figure 3-2. Flow of DMA Controller Operation

At completion of the DMA process, the 82380 will remove the bus hold request. At this time the 82380 becomes a slave again, and the host returns to being a master. If there are other DMA channels with requests pending, the controller will again assert the hold request signal and restart the bus arbitration and switching process.

3.2 Interface Signals

There are fourteen control signals dedicated to the DMA process. They include eight DMA Channel Requests (DREQn), three Encoded DMA Acknowledge signals (EDACKn), Processor Hold and Hold Acknowledge (HOLD, HLDA), and End-Of-Process (EOP#). The DREQn inputs and EDACK(0-2) outputs are handshake signals to the devices requiring DMA service. The HOLD output and HLDA input are handshake signals to the host processor. Figure 3-3 shows these signals and how they interconnect between the 82380 DMA Controller, and the Requester and Target devices.

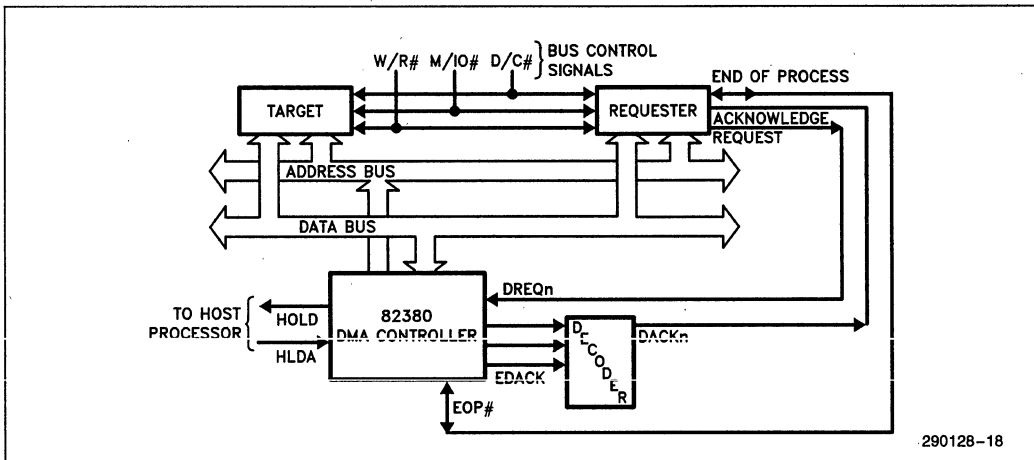


Figure 3-3. Requester, Target, and DMA Controller Interconnection (2-Cycle Configuration)

3.2.1 DREQn and EDACK(0-2)

These signals are the handshake signals between the peripheral and the 82380. When the peripheral requires DMA service, it asserts the DREQn signal of the channel which is programmed to perform the service. The 82380 arbitrates the DREQn against other pending requests and begins the DMA process after finishing other higher priority processes.

When the DMA service for the requested channel is in progress, the EDACK(0-2) signals represent the DMA channel which is accessing the Requester. The 3-bit code on the EDACK(0-2) lines indicates the number of the channel presently being serviced. Table 3-2 shows the encoding of these signals. Note that Channel 4 does not have a corresponding hardware acknowledge.

The DMA acknowledge (EDACK) signals indicate the active channel only during DMA accesses to the Requester. During accesses to the Target, EDACK(0-2) has the idle code (100). EDACK(0-2) can thus be used to select a Requester device during a transfer.

Table 3-2. EDACK Encoding During a DMA Transfer

EDACK2	EDACK1	EDACK0	Active Channel
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	Target Access
1	0	1	5
1	1	0	6
1	1	1	7

DREQn can be programmed as either an Asynchronous or Synchronous input. See section 3.4.1 for details on synchronous versus asynchronous operation of this pin.

The EDACKn signals are always active. They either indicate 'no acknowledge' or they indicate a bus access to the requester. The acknowledge code is either 100, for an idle DMA or during a DMA access to the Target, or 'n' during a Requester access, where n is the binary value representing the channel. A simple 3-line to 8-line decoder can be used to provide discrete acknowledge signals for the peripherals.

3.2.2 HOLD and HLDA

The Hold Request (HOLD) and Hold Acknowledge (HLDA) signals are the handshake signals between

the DMA Controller and the host processor. HOLD is an output from the 82380 and HLDA is an input. HOLD is asserted by the DMA Controller when there is a pending DMA request, thus requesting the processor to give up control of the bus so the DMA process can take place. The 80386 responds by asserting HLDA when it is ready to relinquish control of the bus.

The 82380 will begin operations on the bus one clock cycle after the HLDA signal goes active. For this reason, other devices on the bus should be in the slave mode when HLDA is active.

HOLD and HLDA should not be used to gate or select peripherals requesting DMA service. This is because of the use of DMA-like operations by the DRAM Refresh Controller. The Refresh Controller is arbitrated with the DMA Controller for control of the bus, and refresh cycles have the highest priority. A refresh cycle will take place between DMA cycles without relinquishing bus control. See section 3.4.3 for a more detailed discussion of the interaction between the DMA Controller and the DRAM Refresh Controller.

3.2.3 EOP#

EOP# is a bi-directional signal used to indicate the end of a DMA process. The 82380 activates this as an output during the T2 states of the last Requester bus cycle for which a channel is programmed to execute. The Requester should respond by either withdrawing its DMA request, or interrupting the host processor to indicate that the channel needs to be programmed with a new buffer. As an input, this signal is used to tell the DMA Controller that the peripheral being serviced does not require any more data to be transferred. This indicates that the current buffer is to be terminated.

EOP# can be programmed as either an Asynchronous or a Synchronous input. See section 3.4.1 for details on synchronous versus asynchronous operation of this pin.

3.3 Modes of Operation

The 82380 DMA Controller has many independent operating functions. When designing peripheral interfaces for the 82380 DMA Controller, all of the functions or modes must be considered. All of the channels are independent of each other (except in priority of operation) and can operate in any of the modes. Many of the operating modes, though independently programmable, affect the operation of other modes. Because of the large number of com-

binations possible, each programmable mode is discussed here with its affects on the operation of other modes. The entire list of possible combinations will not be presented.

Table 3-1 shows the categories of DMA features available in the 82380. Each of the five major categories is independent of the others. The sub-categories are the available modes within the major function or mode category. The following sections explain each mode or function and its relation to other features.

Table 3-1. DMA Operating Modes

I. Target/Requester Definition

- a. Data Transfer Direction
- b. Device Type
- c. Increment/Decrement/Hold

II. Buffer Processes

- a. Single Buffer Process
- b. Buffer Auto-Initialize Process
- c. Buffer Chaining Process

III. Data Transfer/Handshake Modes

- a. Single Transfer Mode
- b. Demand Transfer Mode
- c. Block Transfer Mode
- d. Cascade Mode

IV. Priority Arbitration

- a. Fixed
- b. Rotating
- c. Programmable Fixed

V. Bus Operation

- a. Fly-By (Single-Cycle)/Two-Cycle
- b. Data Path Width
- c. Read, Write, or Verify Cycles

3.3.1 TARGET/REQUESTER DEFINITION

All DMA transfers involve three devices: the DMA Controller, the Requester, and the Target. Since the devices to be accessed by the DMA Controller vary widely, the operating characteristics of the DMA Controller must be tailored to the Requester and Target devices.

The Requester can be defined as either the source or the destination of the data to be transferred. This is done by specifying a Write or a Read transfer, respectively. In a Read transfer, the Target is the data source and the Requester is the destination for

the data. In a Write transfer, the Requester is the source and the Target in the destination.

The Requester and Target addresses can each be independently programmed to be incremented, decremented, or held constant. As an example, the 82380 is capable of reversing a string or data by having a Requester address increment and the Target address decrement in a memory-to-memory transfer.

3.3.2 BUFFER TRANSFER PROCESSES

The 82380 DMA Controller allows three programmable Buffer Transfer Processes. These processes define the logical way in which a buffer of data is accessed by the DMA.

The three Buffer Transfer Processes include the Single Buffer Process, the Buffer Auto-Initialize Process, and the Buffer Chaining Process. These processes require special programming considerations. See the DMA Programming section for more details on setting up the Buffer Transfer Processes.

Single Buffer Process

The Single Buffer Process allows the DMA channel to transfer only one buffer of data. When the buffer has been completely transferred (Current Byte Count decremented past zero or EOP# input active), the DMA process ends and the channel becomes idle. In order for that channel to be used again, it must be reprogrammed.

The single Buffer Process is usually used when the amount of data to be transferred is known exactly, and it is also known that there is not likely to be any data to follow before the operating system can reprogram the channel.

Buffer Auto-Initialize Process

The Buffer Auto-Initialize Process allows multiple groups of data to be transferred to or from a single buffer. This process does not require reprogramming. The Current Registers are automatically reprogrammed from the Base Registers when the current process is terminated, either by an expired Byte Count or by an external EOP# signal. The data transferred will always be between the same Target and Requester.

The auto-initialization/process-execution cycle is repeated, with a HOLD/HLDA re-arbitration, until the channel is either disabled or re-programmed.

Buffer Chaining Process

The Buffer Chaining Process is useful for transferring large quantities of data into non-contiguous buffer areas. In this process, a single channel is used to process data from several buffers, while having to program the channel only once. Each new buffer is programmed in a pipelined operation that provides the new buffer information while the old buffer is being processed. The chain is created by loading new buffer information while the 82380 DMA Controller is processing the Current Buffer. When the Current Buffer expires, the 82380 DMA Controller automatically restarts the channel using the new buffer information.

Loading the new buffer information is done by an interrupt routine which is requested by the 82380. Interrupt Request 1 (IRQ1) is tied internally to the 82380 DMA Controller for this purpose. IRQ1 is generated by the 82380 when the new buffer information is loaded into the channel's Current Registers, leaving the Base Registers 'empty'. The interrupt service routine loads new buffer information into the Base Registers. The host processor is required to load the information for another buffer before the current Byte Count expires. The process repeats until the host programs the channel back to single buffer operation, or until the channel runs out of buffers.

The channel runs out of buffers when the Current Buffer expires and the Base Registers have not yet been loaded with new buffer information. When this occurs, the channel must be reprogrammed.

If an external EOP# is encountered while executing a Buffer Chaining Process, the current buffer is considered expired and the new buffer information is loaded into the Current Registers. If the Base Registers are 'empty', the chain is terminated.

The channel uses the Base Target Address Register as an indicator of whether or not the Base Registers are full. When the most significant byte of the Base Target Register is loaded, the channel considers all of the Base Registers loaded, and removes the interrupt request. This requires that the other Base Registers (Base Requester Address, Last Byte Count) must be loaded before the Base Target Address Register. The reason for implementing the re-

loading process this way is that, for most applications, the Byte Count and the Requester will not change from one buffer to the next, and therefore do not need to be reprogrammed. The details of programming the channel for the Buffer Chaining Process can be found in the section of DMA programming.

3.3.3 DATA TRANSFER MODES

Three Data Transfer modes are available in the 82380 DMA Controller. They are the Single Transfer, Block Transfer, and Demand Transfer Modes. These transfer modes can be used in conjunction with any one of three Buffer Transfer modes: Single Buffer, Auto-Initialized Buffer, and Buffer Chaining. Any Data Transfer Modes can be used under any of the Buffer Transfer Modes. These modes are independently available for all DMA channels.

Different devices being serviced by the DMA Controller require different handshaking sequences for data transfers to take place. Three handshaking modes are available on the 82380, giving the designer the opportunity to use the DMA Controller as efficiently as possible. The speed at which data can be presented or read by a device can affect the way a DMA controller uses the host's bus, thereby affecting not only data throughput during the DMA process, but also affecting the host's performance by limiting its access to the bus.

Single Transfer Mode

In the Single Transfer Mode, one data transfer to or from the Requester is performed by the DMA Controller at a time. The DREQn input is arbitrated and the HOLD/HLDA sequence is executed for each transfer. Transfers continue in this manner until the Byte Count expires, or until EOP# is sampled active. If the DREQn input is held active continuously, the entire DREQ-HOLD-HLDA-DACK sequence is repeated over and over until the programmed number of bytes has been transferred. Bus control is released to the host between each transfer. Figure 3-4 shows the logical flow of events which make up a buffer transfer using the Single Transfer Mode. Refer to section 3.4 for an explanation of the bus control arbitration procedure.

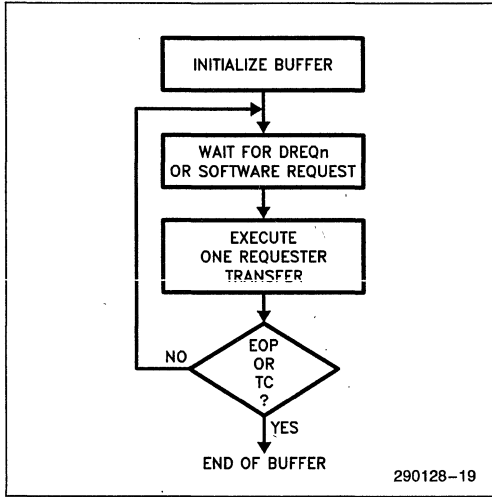


Figure 3-4. Buffer Transfer in Single Transfer Mode

The Single Transfer Mode is used for devices which require complete handshake cycles with each data access. Data is transferred to or from the Requester only when the Requester is ready to perform the transfer. Each transfer requires the entire DREQ-HOLD-HLDA-DACK handshake cycle. Figure 3-5 shows the timing of the Single Transfer Mode cycles.

Block Transfer Mode

In the Block Transfer Mode, the DMA process is initiated by a DMA request and continues until the Byte count expires, or until EOP# is activated by the Requester. The DREQn signal need only be held active until the first Requester access. Only a refresh cycle will interrupt the block transfer process.

Figure 3-6 illustrates the operation of the DMA during the Block Transfer Mode. Figure 3-7 shows the timing of the handshake signals during Block Mode Transfers.

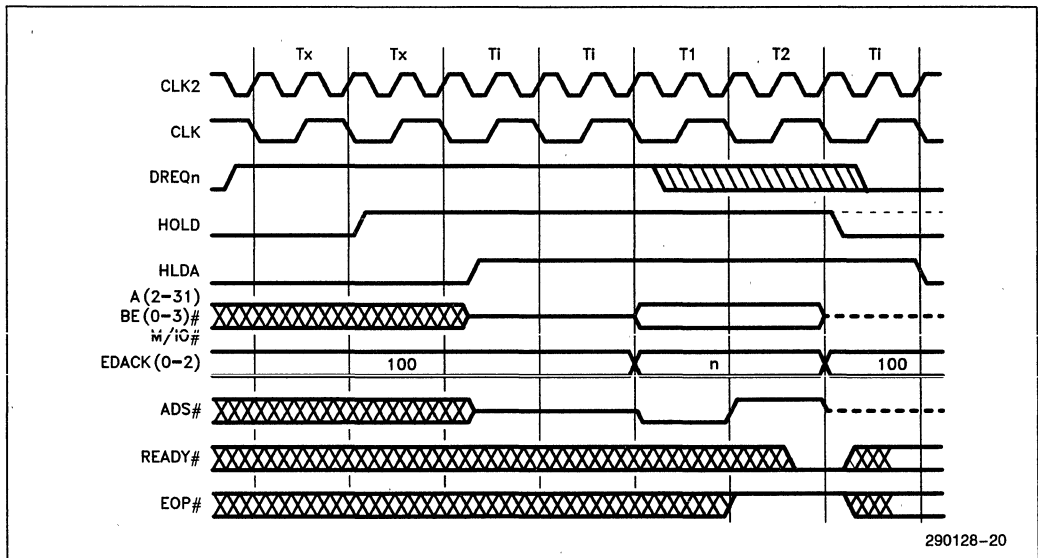


Figure 3-5. DMA Single Transfer Mode

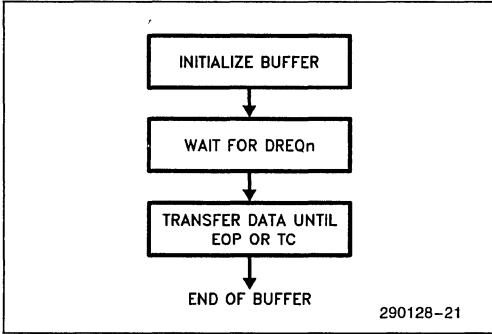


Figure 3-6. Buffer Transfer in Block Transfer Mode

Demand Transfer Mode

The Demand Transfer Mode provides the most flexible handshaking procedures during the DMA process. A Demand Transfer is initiated by a DMA request. The process continues until the Byte Count expires, or an external EOP# is encountered. If the device being serviced (Requester) desires, it can interrupt the DMA process by de-activating the DREQn line. Action is taken on the condition of DREQn during Requester accesses only. The access during which DREQn is sampled inactive is the last Requester access which will be performed during the current transfer. Figure 3-8 shows the flow of events during the transfer of a buffer in the Demand Mode.

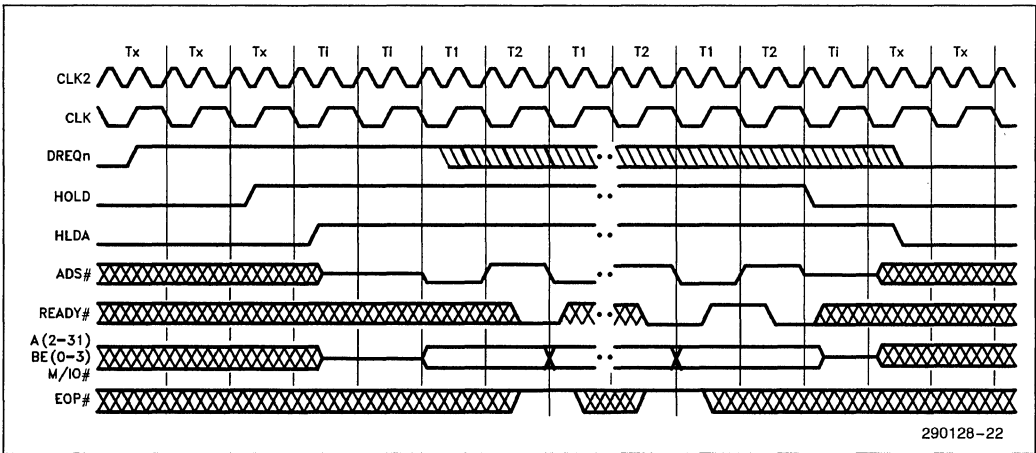


Figure 3-7. Block Mode Transfers

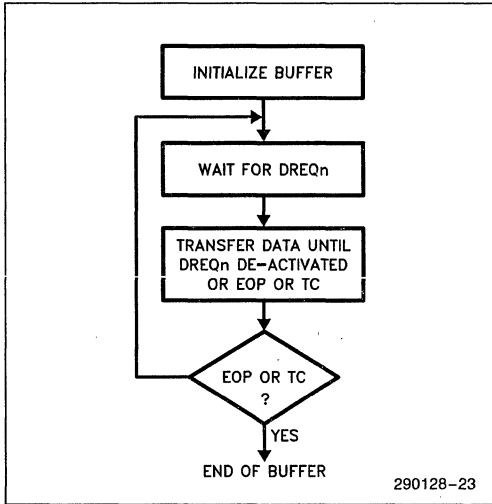


Figure 3-8. Buffer Transfer in Demand Transfer Mode

When the DREQn line goes inactive, the DMA controller will complete the current transfer, including any necessary accesses to the Target, and relinquish control of the bus to the host. The current process information is saved (byte count, Requester and Target addresses, and Temporary Register).

The Requester can restart the transfer process by reasserting DREQn. The 82380 will arbitrate the request with other pending requests and begin the process where it left off. Figure 3-9 shows the timing of handshake signals during Demand Transfer Mode operation.

Using the Demand Transfer Mode allows peripherals to access memory in small, irregular bursts without wasting bus control time. The 82380 is designed to give the best possible bus control latency in the Demand Transfer Mode. Bus control latency is defined here as the time from the last active bus cycle of the previous bus master to the first active bus cycle of the new bus master. The 82380 DMA Controller will perform its first bus access cycle two bus states after HLDA goes active. In the typical configuration, bus control is returned to the host one bus state after the DREQn goes inactive.

There are two cases where there may be more than one bus state of bus control latency at the end of a transfer. The first is at the end of an Auto-Initialize process, and the second is at the end of a process where the source is the Requester and Two-Cycle transfers are used.

When a Buffer Auto-Initialize Process is complete, the 82380 requires seven bus states to reload the

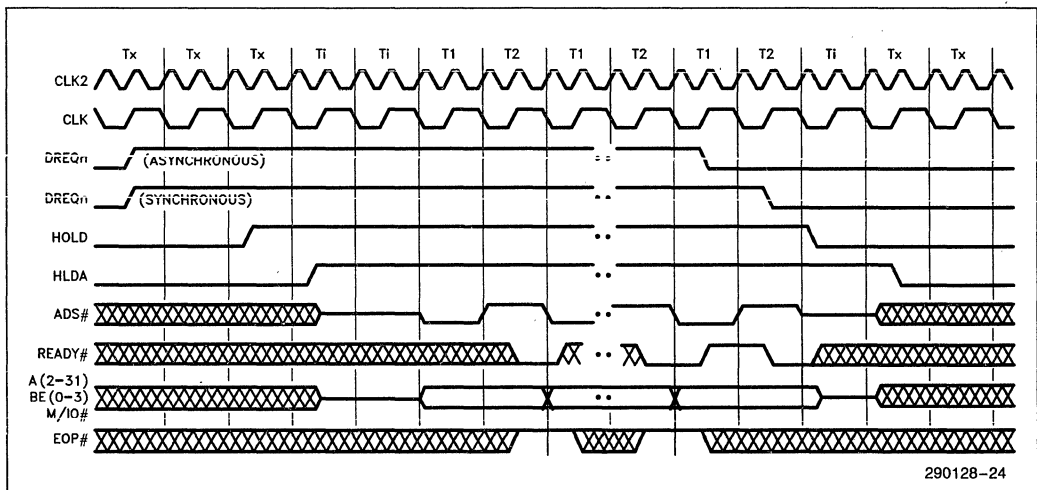


Figure 3-9. Demand Mode Transfers

Current Registers from the Base Registers of the Auto-Initialized channel. The reloading is done while the 82380 is still the bus master so that it is prepared to service the channel immediately after relinquishing the bus, if necessary.

In the case where the Requester is the source, and Two-Cycle transfers are being used, there are two extra idle states at the end of the transfer process. This occurs due to housekeeping in the DMA's internal pipeline. These two idle states are present only after the very last Requester access, before the DMA Controller de-activates the HOLD signal.

3.3.4 CHANNEL PRIORITY ARBITRATION

DMA channel priority can be programmed into one of two arbitration methods: Fixed or Rotating. The four lower DMA channels and the four upper DMA channels operate as if they were two separate DMA controllers operating in cascade. The lower group of four channels (0-3) is always prioritized between channels 7 and 4 of the upper group of channels (4-7). Figure 3-10 shows a pictorial representation of the priority grouping.

The priority can thus be set up as rotating for one group of channels and fixed for the other, or any other combination. While in Fixed Priority, the programmer can also specify which channel has the lowest priority.

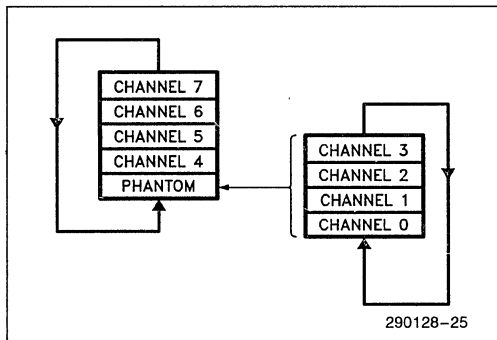


Figure 3-10. DMA Priority Grouping

The 82380 DMA Controller defaults to Fixed Priority. Channel 0 has the highest priority, then 1, 2, 3, 4, 5, 6, 7. Channel 7 has the lowest priority. Any time the DMA Controller arbitrates DMA requests, the requesting channel with the highest priority will be serviced next.

Fixed Priority can be entered into at any time by a software command. The priority levels in effect

after the mode switch are determined by the current setting of the Programmable Priority.

Programmable Priority is available for fixing the priority of the DMA channels within a group to levels other than the default. Through a software command, the channel to have the lowest priority in a group can be specified. Each of the two groups of four channels can have the priority fixed in this way. The other channels in the group will follow the natural Fixed Priority sequence. This mode affects only the priority levels while operating with Fixed Priority.

For example, if channel 2 is programmed to have the lowest priority in its group, channel 3 has the highest priority. In descending order, the other channels would have the following priority: (3, 0, 1, 2), 4, 5, 6, 7 (channel 2 lowest, channel 3 highest). If the upper group were programmed to have channel 5 as the lowest priority channel, the priority would be (again, highest to lowest): 6, 7, (3, 0, 1, 2), 4, 5. Figure 3-11 shows this example pictorially. The lower group is always prioritized as a fifth channel of the upper group (between channels 4 and 7).

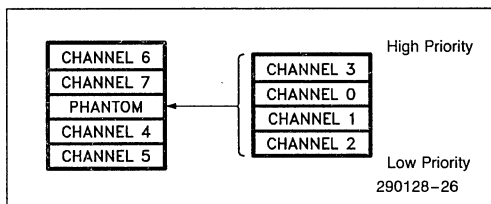


Figure 3-11. Example of Programmed Priority

The DMA Controller will only accept Programmable Priority commands while the addressed group is operating in Fixed Priority. Switching from Fixed to Rotating Priority preserves the current priority levels. Switching from Rotating to Fixed Priority returns the priority levels to those which were last programmed by use of Programmable Priority.

Rotating Priority allows the devices using DMA to share the system bus more evenly. An individual channel does not retain highest priority after being serviced, priority is passed to the next highest priority channel in the group. The channel which was most recently serviced inherits the lowest priority. This rotation occurs each time a channel is serviced. Figure 3-12 shows the sequence of events as priority is passed between channels. Note that the lower group rotates within the upper group, and that servicing a channel within the lower group causes rotation within the group as well as rotation of the upper group.

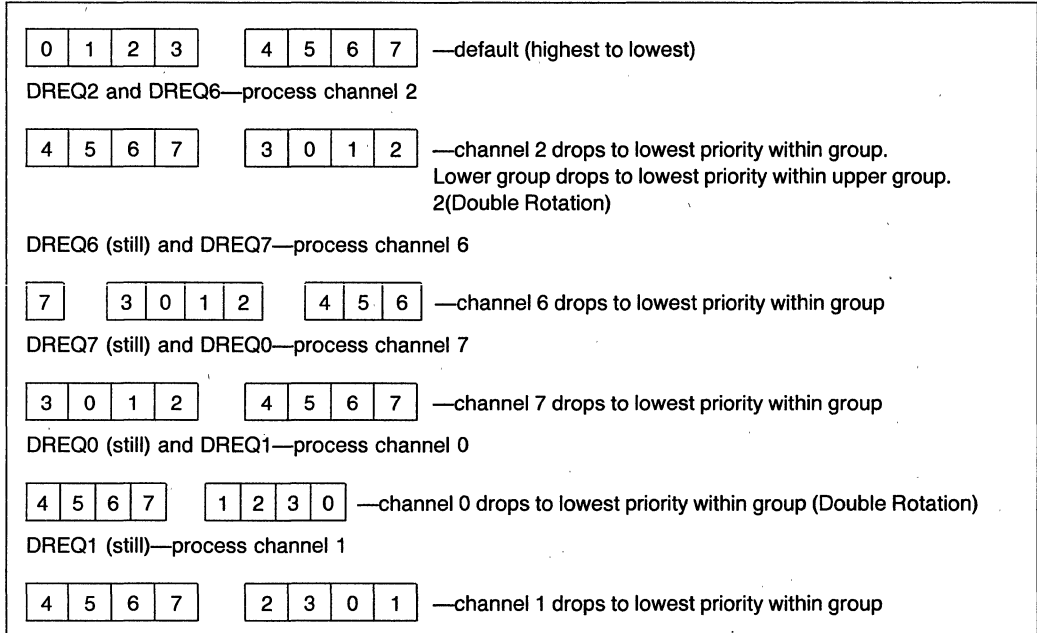


Figure 3-12. Rotating Channel Priority. Lower and Upper groups are programmed for the Rotating Priority Mode.

3.3.5 COMBINING PRIORITY MODES

Since the DMA Controller operates as two four-channel controllers in cascade, the overall priority scheme of all eight channels can take on a variety of forms. There are four possible combinations of prior-

ity modes between the two groups of channels: Fixed Priority only (default), Fixed Priority upper group/Rotating Priority lower group, Rotating Priority upper group/Fixed Priority lower group, and Rotating Priority only. Figure 3-13 illustrates the operation of the two combined priority methods.

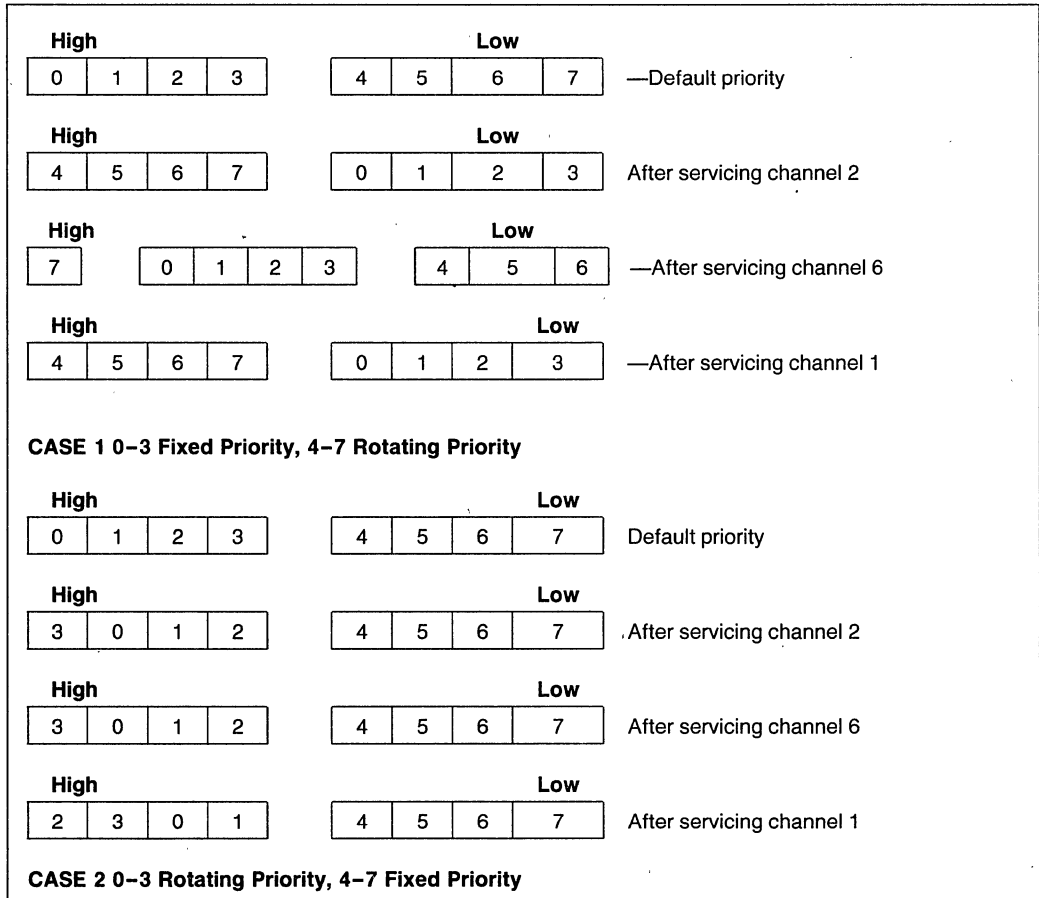


Figure 3-13. Combining Priority Modes

3.3.6 BUS OPERATION

Data may be transferred by the DMA Controller using two different bus cycle operations: Fly-By (one-cycle) and Two-Cycle. These bus handshake methods are selectable independently for each channel through a command register. Device data path widths are independently programmable for both Target and Requester. Also selectable through software is the direction of data transfer. All of these parameters affect the operation of the 82380 on a bus-cycle by bus-cycle basis.

3.3.6.1 Fly-By Transfers

The Fly-By Transfer Mode is the fastest and most efficient way to use the 82380 DMA Controller to transfer data. In this method of transfer, the data is written to the destination device at the same time it is read from the source. Only one bus cycle is used to accomplish the transfer.

In the Fly-By Mode, the DMA acknowledge signal is used to select the Requester. The DMA Controller simultaneously places the address of the Target on the address bus. The state of M/IO# and W/R# during the Fly-By transfer cycle indicate the type of Target and whether the target is being written to or read from. The Target's Bus Size is used as an incrementer for the Byte Count. The Requester address registers are ignored during Fly-By transfers.

Note that memory-to-memory transfers cannot be done using the Fly-By Mode. Only one memory or I/O address is generated by the DMA Controller at a time during Fly-By transfers. Only one of the devices being accessed can be selected by an address. Also, the Fly-By method of data transfer limits the hardware to accesses of devices with the same data bus width. The Temporary Registers are not affected in the Fly-By Mode.

Fly-By transfers also require that the data paths of the Target and Requester be directly connected. This requires that successive Fly-By accesses be to doubleword boundaries, or that the Requester be capable of switching its connections to the data bus.

3.3.6.2 Two-Cycle Transfers

Two-Cycle transfers can also be performed by the 82380 DMA Controller. These transfers require at least two bus cycles to execute. The data being transferred is read into the DMA Controller's Temporary Register during the first bus cycle(s). The second bus cycle is used to write the data from the Temporary Register to the destination.

If the addresses of the data being transferred are not word or doubleword aligned, the 82380 will recognize the situation and read and write the data in groups of bytes, placing them always at the proper destination. This process of collecting the desired bytes and putting them together is called 'byte assembly'. The reverse process (reading from aligned locations and writing to non-aligned locations) is called 'byte disassembly'.

The assembly/disassembly process takes place transparent to the software, but can only be done while using the Two-Cycle transfer method. The 82380 will always perform the assembly/disassembly process as necessary for the current data transfer. Any data path widths for either the Requester or Target can be used in the Two-Cycle Mode. This is very convenient for interfacing existing 8- and 16-bit peripherals to the 80386's 32-bit bus.

The 82380 DMA Controller always attempts to fill the Temporary Register from the source before writing any data to the destination. If the process is terminated before the Temporary Register is filled (TC or EOP#), the 82380 will write the partial data to the destination. If a process is temporarily suspended (such as when DREQn is de-activated during a demand transfer), the contents of a partially filled Temporary Register will be stored within the 82380 until the process is restarted.

For example, if the source is specified as an 8-bit device and the destination as a 32-bit device, there will be four reads as necessary from the 8-bit source to fill the Temporary Register. Then the 82380 will write the 32-bit contents to the destination. This cycle will repeat until the process is terminated or suspended.

Note that for a Single-Cycle transfer mode of operation (see section 3.3.3), the internal circuitry of the DMA Controller actually executes single transfers by removing the DREQ from the internal arbitration. Thus single transfers from an 8-bit requester to a 32-bit target will consist of four complete and independent 8-bit requester cycles, between which bus control is released and re-requested. Finally, the 32-bit data will be transferred to the target device from the temporary register before the fifth requester cycle.

With Two-Cycle transfers, the devices that the 82380 accesses can reside at any address within I/O or memory space. The device must be able to decode the byte-enables (BEN#). Also, if the device cannot accept data in byte quantities, the programmer must take care not to allow the DMA Controller to access the device on any address other than the device boundary.

3.3.6.3 Data Path Width and Data Transfer Rate Considerations

The number of bus cycles used to transfer a single 'word' of data is affected by whether the Two-Cycle or the Fly-By (Single-Cycle) transfer method is used.

The number of bus cycles used to transfer data directly affects the data transfer rate. Inefficient use of bus cycles will decrease the effective data transfer rate that can be obtained. Generally, the data transfer rate is halved by using Two-Cycle transfers instead of Fly-By transfers.

The choice of data path widths of both Target and Requester affects the data transfer rate also. During each bus cycle, the largest pieces of data possible should be transferred.

The data path width of the devices to be accessed must be programmed into the DMA controller. The 82380 defaults after reset to 8-bit-to-8-bit data transfers, but the Target and Requester can have different data path widths, independent of each other and independent of the other channels. Since this is a software programmable function, more discussion of the uses of this feature are found in the section on programming.

3.3.6.4 Read, Write, and Verify Cycles

Three different bus cycle types may be used in a data transfer. They are the Read, Write, and Verify cycles. These cycle types dictate the way in which the 82380 operates on the data to be transferred.

A Read Cycle transfers data from the Target to the Requester. A Write Cycle transfers data from the Requester to the target. In a Fly-By transfer, the address and bus status signals indicate the access (read or write) to the Target; the access to the Requester is assumed to be the opposite.

The Verify Cycle is used to perform a data read only. No write access is indicated or assumed in a Verify Cycle. The Verify Cycle is useful for validating block fill operations. An external comparator must be provided to do any comparisons on the data read.

3.4 Bus Arbitration and Handshaking

Figure 3-14 shows the flow of events in the DMA request arbitration process. The arbitration sequence

starts when the Requester asserts a DREQn (or DMA service is requested by software). Figure 3-15 shows the timing of the sequence of events following a DMA request. This sequence is executed for each channel that is activated. The DREQn signal can be replaced by a software DMA channel request with no change in the sequence.

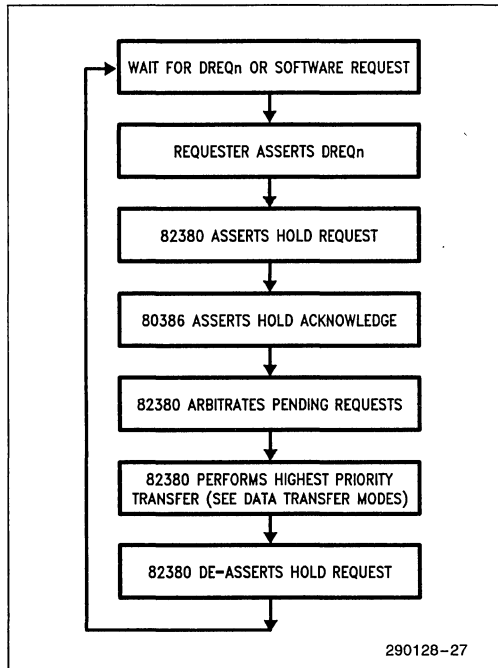


Figure 3-14. Bus Arbitration and DMA Sequence

After the Requester asserts the service request, the 82380 will request control of the bus via the HOLD signal. The 82380 will always assert the HOLD signal one bus state after the service request is asserted. The 80386 responds by asserting the HLDA signal, thus releasing control of the bus to the 82380 DMA Controller.

Priority of pending DMA service requests is arbitrated during the first state after HLDA is asserted by the 80386. The next state will be the beginning of the first transfer access of the highest priority process.

When the 82380 DMA Controller is finished with its current bus activity, it returns control of the bus to the host processor. This is done by driving the HOLD signal inactive. The 82380 does not drive any address or data bus signals after HOLD goes low. It enters the Slave Mode until another DMA process is requested. The processor acknowledges that it has regained control of the bus by forcing the HLDA signal inactive. Note that the 82380's DMA Controller will not re-request control of the bus until the entire HOLD/HLDA handshake sequence is complete.

The 82380 DMA Controller will terminate a current DMA process for one of three reasons: expired byte count, end-of-process command (EOP# activated) from a peripheral, or de-activated DMA request signal. In each case, the controller will de-assert HOLD immediately after completing the data transfer in progress. These three methods of process termination are illustrated in Figures 3-16, 3-19, and 3-18, respectively.

An expired byte count indicates that the current process is complete as programmed and the channel has no further transfers to process. The channel must be restarted according to the currently programmed Buffer Transfer Mode, or reprogrammed completely, including a new Buffer Transfer Mode.

If the peripheral activates the EOP# signal, it is indicating that it will not accept or deliver any more data for the current buffer. The 82380 DMA Controller considers this as a completion of the channel's current process and interprets the condition the same way as if the byte count expired.

The action taken by the 82380 DMA Controller in response to a de-activated DREQn signal depends on the Data Transfer Mode of the channel. In the Demand Mode, data transfers will take place as long as the DREQn is active and the byte count has not expired. In the Block Mode, the controller will complete the entire block transfer without relinquishing

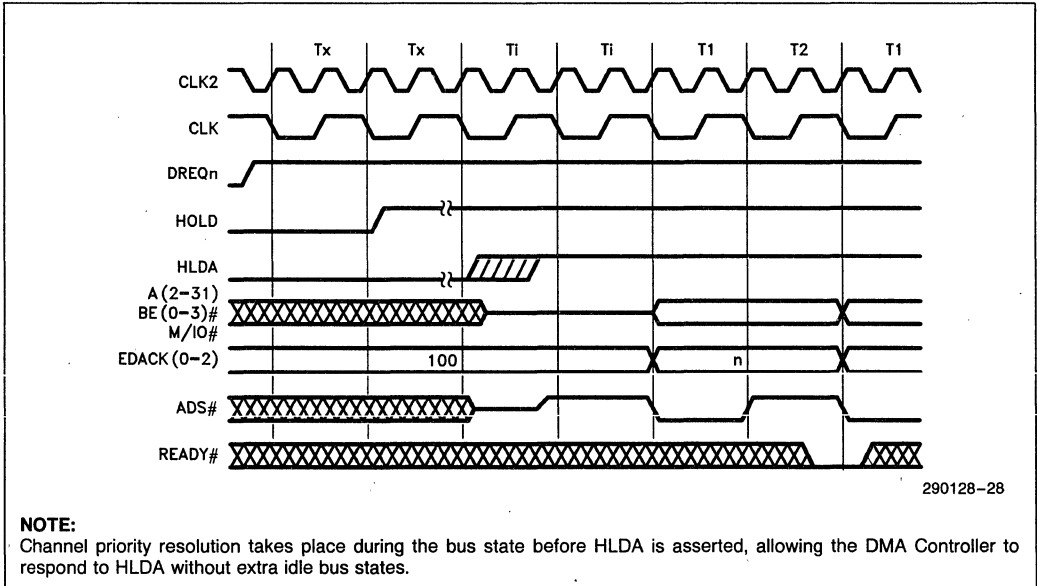


Figure 3-15. Beginning of a DMA process

the bus, even if DREQn goes inactive before the transfer is complete. In the Single Mode, the controller will execute single data transfers, relinquishing the bus between each transfer, as long as DREQn is active.

in Figure 3-16. The condition of DREQn is ignored until after the process is terminated. If the channel is programmed to auto-initialize, HOLD will be held active for an additional seven clock cycles while the auto-initialization takes place.

Normal termination of a DMA process due to expiration of the byte count (Terminal Count-TC) is shown

Table 3-3 shows the DMA channel activity due to EOP# or Byte Count expiring (Terminal Count).

Buffer Process:	Single or Chaining-Base Empty		Auto-Initialize		Chaining-Base Loaded	
	True	X	True	X	True	X
Event						
Terminal Count	True	X	True	X	True	X
EOP# Input	X	0	X	0	X	0
Results						
Current Registers	—	—	Load	Load	Load	Load
Channel Mask	Set	Set	—	—	—	—
EOP# Output	0	X	0	X	1	X
Terminal Count Status	Set	Set	Set	Set	—	—
Software Request	CLR	CLR	CLR	CLR	—	—

Table 3-3. DMA Channel Activity Due to Terminal Count or External EOP#

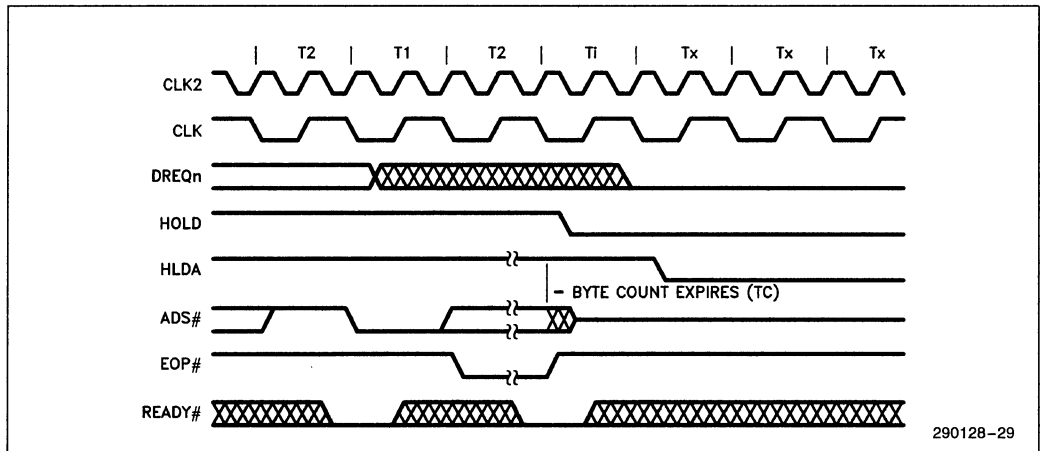


Figure 3-16. Termination of a DMA Process Due to Expiration of Current Byte Count

The 82380 always relinquishes control of the bus between channel services. This allows the hardware designer the flexibility to externally arbitrate bus hold requests, if desired. If another DMA request is pending when a higher priority channel service is completed, the 82380 will relinquish the bus until the hold acknowledge is inactive. One bus state after the HLDA signal goes inactive, the 82380 will assert HOLD again. This is illustrated in Figure 3-17.

3.4.1 SYNCHRONOUS AND ASYNCHRONOUS SAMPLING OF DREQn AND EOP#

As an indicator that a DMA service is to be started, DREQn is always sampled asynchronously. It is sampled at the beginning of a bus state and acted upon at the end of the state. Figure 3-15 illustrates the start of a DMA process due to a DREQn input.

The DREQn and EOP# inputs can be programmed to be sampled either synchronously or asynchronously to signal the end of a transfer.

The synchronous mode affords the Requester one bus state of extra time to react to an access. This means the Requester can terminate a process on the current access, without losing any data. The asynchronous mode requires that the input signal be presented prior to the beginning of the last state of the Requester access.

The timing relationships of the DREQn and EOP# signals to the termination of a DMA transfer are shown in Figures 3-18 and 3-19. Figure 3-18 shows the termination of a DMA transfer due to inactive DREQn. Figure 3-19 shows the termination of a DMA process due to an active EOP# input.

In the Synchronous Mode, DREQn and EOP# are sampled at the end of the last state of every Requester data transfer cycle. If EOP# is active or DREQn is inactive at this time, the 82380 recognizes this access to the Requester as the last transfer. At this point, the 82380 completes the transfer in progress, if necessary, and returns bus control to the host.

In the asynchronous mode, the inputs are sampled at the beginning of every state of a Requester access. The 82380 waits until the end of the state to act on the input.

DREQn and EOP# are sampled at the latest possible time when the 82380 can determine if another transfer is required. In the Synchronous Mode, DREQn and EOP# are sampled on the trailing edge of the last bus state before another data access cycle begins. The Asynchronous Mode requires that the signals be valid one clock cycle earlier.

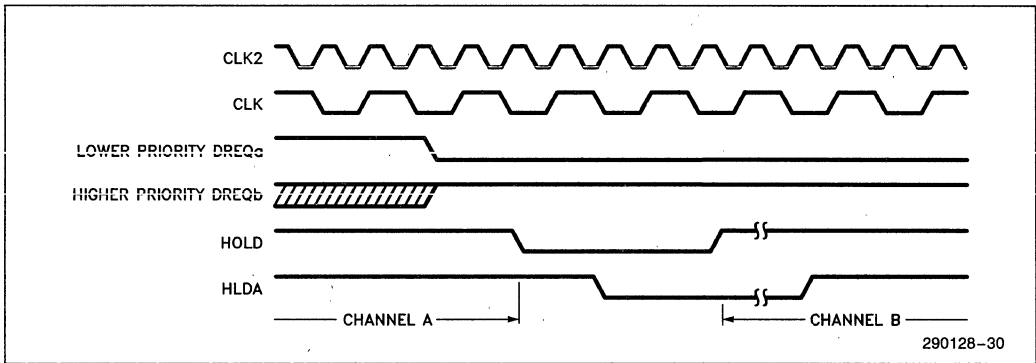


Figure 3-17. Switching between Active DMA Channels

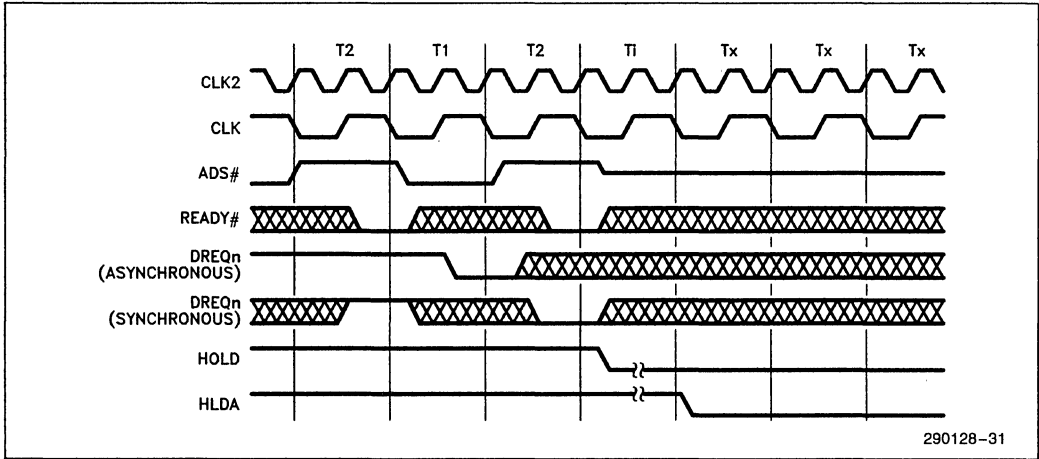


Figure 3-18. Termination of a DMA Process Due to De-Asserting DREQn

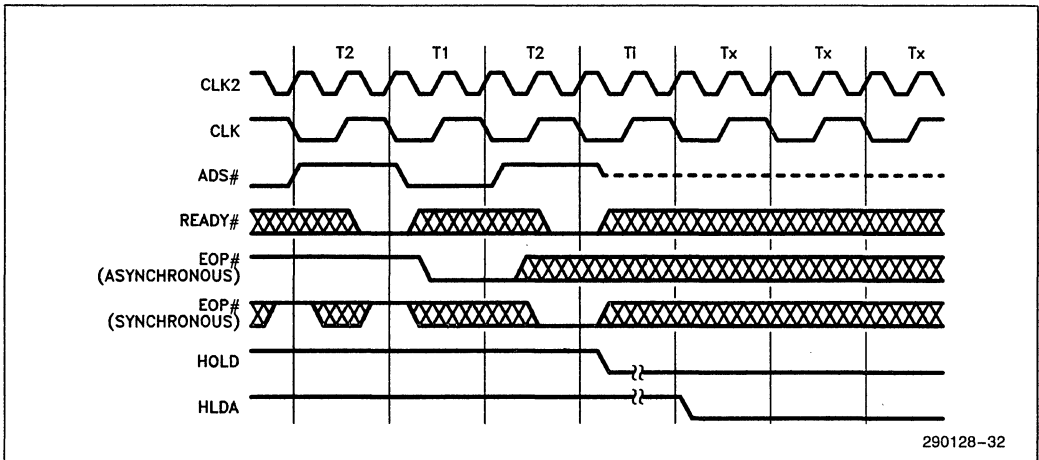


Figure 3-19. Termination of a DMA Process Due to an External EOP#

While in the Pipeline Mode, if the NA# signal is sampled active during a transfer, the end of the state where NA# was sampled active is when the 82380 decides whether to commit to another transfer. The device must de-assert DREQn or assert EOP# before NA# is asserted, otherwise the 82380 will commit to another, possibly undesired, transfer.

Synchronous DREQn and EOP# sampling allows the peripheral to prevent the next transfer from occurring by de-activating DREQn or asserting EOP# during the current Requester access, before the 82380 DMA Controller commits itself to another transfer. The DMA Controller will not perform the next transfer if it has not already begun the bus cycle. Asynchronous sampling allows less stringent timing requirements than the Synchronous Mode, but requires that the DREQn signal be valid at the beginning of the next to last bus state of the current Requester access.

Using the Asynchronous Mode with zero wait states can be very difficult. Since the addresses and control signals are driven by the 82380 near half-way

through the first bus state of a transfer, and the Asynchronous Mode requires that DREQn be active before the end of the state, the peripheral being accessed is required to present DREQn only a few nanoseconds after the control information is available. This means that the peripheral's control logic must be extremely fast (practically non-causal). An alternative is the Synchronous Mode.

3.4.2 ARBITRATION OF CASCADED MASTER REQUESTS

The Cascade Mode allows another DMA-type device to share the bus by arbitrating its bus accesses with the 82380's. Seven of the eight DMA channels (0-3 and 5-7) can be connected to a cascaded device. The cascaded device requests bus control through the DREQn line of the channel which is programmed to operate in Cascade Mode. Bus hold acknowledge is signaled to the cascaded device through the EDACK lines. When the EDACK lines are active with the code for the requested cascade channel, the bus is available to the cascaded master device.

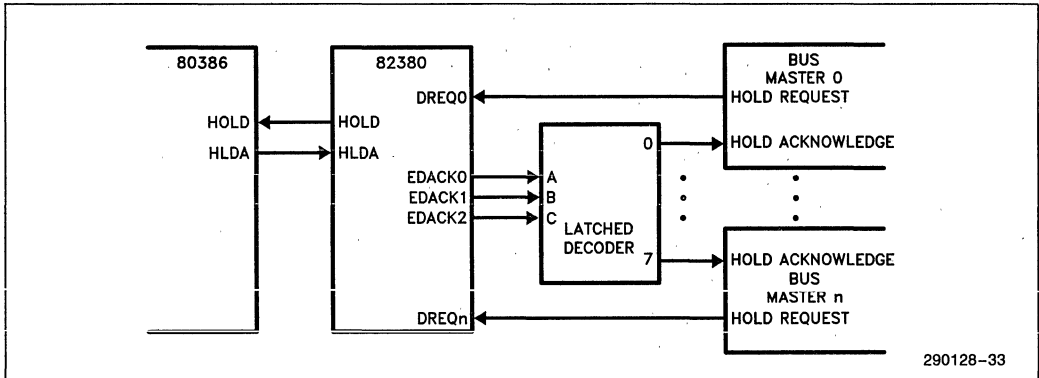


Figure 3-20. Cascaded Bus Master

A Cascade cycle begins the same way a regular DMA cycle begins. The requesting bus master asserts the DREQn line on the 82380. This bus control request is arbitrated as any other DMA request would be. If any channel receives a DMA request, the 82380 requests control of the bus. When the host acknowledges that it has released bus control, the 82380 acknowledges to the requesting master that it may access the bus. The 82380 enters an idle state until the new master relinquishes control.

A cascade cycle will be terminated by one of two events: DREQn going inactive, or HLDA going inactive. The normal way to terminate the cascade cycle

is for the cascaded master to drop the DREQn signal. Figure 3-21 shows the two cascade cycle termination sequences.

The Refresh Controller may interrupt the cascaded master to perform a refresh cycle. If this occurs, the 82380 DMA Controller will de-assert the EDACK signal (hold acknowledge to cascaded master) and wait for the cascaded master to remove its hold request. When the 82380 regains bus control, it will perform the refresh cycle in its normal fashion. After the refresh cycle has been completed, and if the cascaded device has re-asserted its request, the 82380 will return control to the cascaded master which was interrupted.

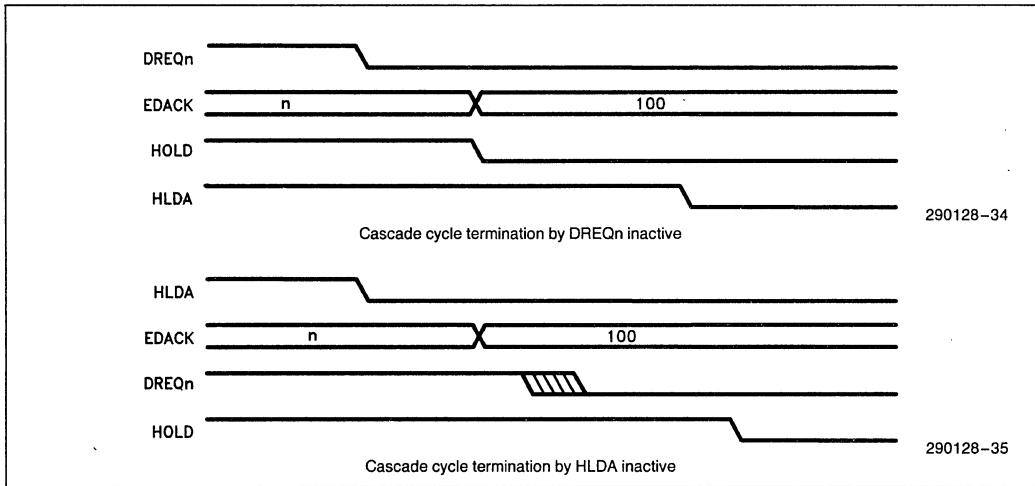


Figure 3-21. Cascade Cycle Termination

The 82380 assumes that it is the only device monitoring the HLDA signal. If the system designer wishes to place other devices on the bus as bus masters, the HLDA from the processor must be intercepted before presenting it to the 82380. Using the Cascade capability of the 82380 DMA Controller offers a much better solution.

3.4.3 ARBITRATION OF REFRESH REQUESTS

The arbitration of refresh requests by the DRAM Refresh Controller is slightly different from normal DMA channel request arbitration. The 82380 DRAM Refresh Controller always has the highest priority of any DMA process. It also can interrupt a process in progress. Two types of processes in progress may be encountered: normal DMA, and bus master cascade.

In the event of a refresh request during a normal DMA process, the DMA Controller will complete the data transfer in progress and then execute the refresh cycle before continuing with the current DMA process. The priority of the interrupted process is not lost. If the data transfer cycle interrupted by the Refresh Controller is the last of a DMA process, the refresh cycle will always be executed before control of the bus is transferred back to the host.

When the Refresh Controller request occurs during a cascade cycle, the Refresh Controller must be assured that the cascaded master device has relinquished control of the bus before it can execute the refresh cycle. To do this, the DMA Controller drops the EDACK signal to the cascaded master and waits for the corresponding DREQn input to go inactive. By dropping the DREQn signal, the cascaded master relinquishes the bus. The Refresh Controller then performs the refresh cycle. Control of the bus is returned to the cascaded master if DREQn returns to an active state before the end of the refresh cycle, otherwise control is passed to the processor and the cascaded master loses its priority.

3.5 DMA Controller Register Overview

The 82380 DMA Controller contains 44 registers which are accessible to the host processor. Twenty-four of these registers contain the device addresses and data counts for the individual DMA channels (three per channel). The remaining registers are control and status registers for initiating and monitoring the operation of the 82380 DMA Controller. Table 3-4 lists the DMA Controller's registers and their accessibility.

Register Name	Access
Control/Status Register—One Each Per Group	
Command Register I	Write Only
Command Register II	Write Only
Mode Register I	Write Only
Mode Register II	Write Only
Software Request Register	Read/Write
Mask Set-Reset Register	Write Only
Mask Read-Write Register	Read/Write
Status Register	Read Only
Bus Size Register	Write Only
Chaining Register	Read/Write
Channel Registers—One Each Per Channel	
Base Target Address	Write Only
Current Target Address	Read Only
Base Requester Address	Write Only
Current Requester Address	Read Only
Base Byte Count	Write Only
Current Byte Count	Read Only

Table 3-4. DMA Controller Registers

3.5.1 CONTROL/STATUS REGISTERS

The following registers are available to the host processor for programming the 82380 DMA Controller into its various modes and for checking the operating status of the DMA processes. Each set of four DMA channels has one of each of these registers associated with it.

Command Register I

Enables or disables the DMA channels as a group. Sets the Priority Mode (Fixed or Rotating) of the group. This write-only register is cleared by a hardware reset, defaulting to all channels enabled and Fixed Priority Mode.

Command Register II

Sets the sampling mode of the DREQn and EOP# inputs. Also sets the lowest priority channel for the group in the Fixed Priority Mode. The functions programmed through Command Register II default after a hardware reset to: asynchronous DREQn and EOP#, and channels 3 and 7 lowest priority.

Mode Register I

Mode Register I is identical in function to the Mode register of the 8237A. It programs the following functions for an individually selected channel:

Type of Transfer—read, write, verify
 Auto—Initialize—enable or disable
 Target Address Count—increment or decrement
 Data Transfer Mode—demand, single, block, cascade

Mode Register I functions default to the following after reset: verify transfer, Auto-Initialize disabled, Increment Target address, Demand Mode.

Mode Register II

Programs the following functions for an individually selected channel:

Target Address Hold—enable or disable
 Requester Address Count—increment or decrement
 Requester Address Hold—enable or disable
 Target Device Type—I/O or Memory
 Requester Device Type—I/O or Memory
 Transfer Cycles—Two-Cycle or Fly-By

Mode Register II functions are defined as follows after a hardware reset: Disable Target Address Hold, Increment Requester Address, Target (and Requester) in memory, Fly-By Transfer Cycles. Note: Requester.Device Type ignored in Fly-By Transfers.

Software Request Register

The DMA Controller can respond to service requests which are initiated by software. Each channel has an internal request status bit associated with it. The host processor can write to this register to set or reset the request bit of a selected channel.

The status of the group's software DMA service requests can be read from this register as well. Each request bit is cleared upon Terminal Count or external EOP#.

The software DMA requests are non-maskable and subject to priority arbitration with all other software and hardware requests. The entire register is cleared by a hardware reset.

Mask Registers

Each channel has associated with it a mask bit which can be set/reset to disable/enable that channel. Two methods are available for setting and clearing the mask bits. The Mask Set/Reset Register is a write-only register which allows the host to select an individual channel and either set or reset the mask bit for that channel only. The Mask Read/Write Register is available for reading the mask bit status and for writing mask bits in groups of four.

The mask bits of a group may be cleared in one step by executing the Clear Mask Command. See the DMA Programming section for details. A hardware reset sets all of the channel mask bits, disabling all channels.

Status Register

The Status register is a read-only register which contains the Terminal Count (TC) and Service Request status for a group. Four bits indicate the TC status and four bits indicate the hardware request status for the four channels in the group. The TC bits are set when the Byte Count expires, or when an external EOP# is asserted. These bits are cleared by reading from the Status Register. The Service Request bit for a channel indicates when there is a hardware DMA request (DREQn) asserted for that channel. When the request has been removed, the bit is cleared.

Bus Size Register

This write-only register is used to define the bus size of the Target and Requester of a selected channel. The bus sizes programmed will be used to dictate the sizes of the data paths accessed when the DMA channel is active. The values programmed into this register affect the operation of the Temporary Register. Any byte-assembly required to make the transfers using the specified data path widths will be done in the Temporary Register. The Bus Size register of the Target is used as an increment/decrement value for the Byte Counter and Target Address when in the Fly-By Mode. Upon reset, all channels default to 8-bit Targets and 8-bit Requesters.

Chaining Register

As a command or write register, the Chaining register is used to enable or disable the Chaining Mode for a selected channel. Chaining can either be disabled or enabled for an individual channel, independently of the Chaining Mode status of other channels. After a hardware reset, all channels default to Chaining disabled.

When read by the host, the Chaining Register provides the status of the Chaining Interrupt of each of the channels. These interrupt status bits are cleared when the new buffer information has been loaded.

3.5.2 CHANNEL REGISTERS

Each channel has three individually programmable registers necessary for the DMA process; they are the Base Byte Count, Base Target Address, and Base Requester Address registers. The 24-bit Base

Byte Count register contains the number of bytes to be transferred by the channel. The 32-bit Base Target Address Register contains the beginning address (memory or I/O) of the Target device. The 32-bit Base Requester Address register contains the base address (memory or I/O) of the device which is to request DMA service.

Three more registers for each DMA channel exist within the DMA Controller which are directly related to the registers mentioned above. These registers contain the current status of the DMA process. They are the Current Byte Count register, the Current Target Address, and the Current Requester Address. It is these registers which are manipulated (incremented, decremented, or held constant) by the 82380 DMA Controller during the DMA process. The Current registers are loaded from the Base registers.

The Base registers are loaded when the host processor writes to the respective channel register addresses. Depending on the mode in which the channel is operating, the Current registers are typically loaded in the same operation. Reading from the channel register addresses yields the contents of the corresponding Current register.

To maintain compatibility with software which accesses an 8237A, a Byte Pointer Flip-Flop is used to control access to the upper and lower bytes of some words of the Channel Registers. These words are accessed as byte pairs at single port addresses. The Byte Pointer Flip-Flop acts as a one-bit pointer which is toggled each time a qualifying Channel Register byte is accessed. It always points to the next logical byte to be accessed of a pair of bytes.

The Channel registers are arranged as pairs of words, each pair with its own port address. Addressing the port with the Byte Pointer Flip-Flop reset accesses the least significant byte of the pair. The most significant byte is accessed when the Byte Pointer is set.

For compatibility with existing 8237A designs, there is one exception to the above statements about the Byte Pointer Flip-Flop. The third byte (bits 16-23) of the Target Address is accessed through its own port address. The Byte Pointer Flip-Flop is not affected by any accesses to this byte.

The upper eight bits of the Byte Count Register are cleared when the least significant byte of the register is loaded. This provides compatibility with software which accesses an 8237A. The 8237A has 16-bit Byte Count Registers.

3.5.3 TEMPORARY REGISTERS

Each channel has a 32-bit Temporary Register used for temporary data storage during two-cycle DMA transfers. It is this register in which any necessary byte assembly and disassembly of non-aligned data is performed. Figure 3-22 shows how a block of data will be moved between memory locations with different boundaries. Note that the order of the data does not change.

SOURCE		DESTINATION	
20H	A	50H	
21H	B	51H	
22H	C	52H	
23H	D	53H	A
24H	E	54H	B
25H	F	55H	C
26H	G	56H	D
27H		57H	E
		58H	F
		59H	G
		5AH	

Target = source = 0000020H
 Requester = destination = 0000053H
 Byte Count = 000006H

Figure 3-22. Transfer of Data between Memory Locations with Different Boundaries. This will be the result, independent of data path width.

If the destination is the Requester and an early process termination has been indicated by the EOP# signal or DREQn inactive in the Demand Mode, the Temporary Register is not affected. If data remains in the Temporary Register due to differences in data path widths of the Target and Requester, it will not be transferred or otherwise lost, but will be stored for later transfer.

If the destination is the Target and the EOP# signal is sensed active during the Requester access of a transfer, the DMA Controller will complete the transfer by sending to the Target whatever information is in the Temporary Register at the time of process termination. This implies that the Target could be accessed with partial data. For this reason it is advisable to have an I/O device designated as a Requester, unless it is capable of handling partial data transfers.

3.6 DMA Controller Programming

Programming a DMA Channel to perform a needed DMA function is in general a four step process. First the global attributes of the DMA Controller are programmed via the two Command Registers. These global attributes include: priority levels, channel group enables, priority mode, and DREQn/EOP# input sampling.

The second step involves setting the operating modes of the particular channel. The Mode Registers are used to define the type of transfer and the handshaking modes. The Bus Size Register and Chaining Register may also need to be programmed in this step.

The third step is setting up the channel is to load the Base Registers in accordance with the needs of the operating modes chosen in step two. The Current Registers are automatically loaded from the Base Registers, if required by the Buffer Transfer Mode in effect. The information loaded and the order in which it is loaded depends on the operating mode. A channel used for cascading, for example, needs no buffer information and this step can be skipped entirely.

The last step is to enable the newly programmed channel using one of the Mask Registers. The channel is then available to perform the desired data transfer. The status of the channel can be observed at any time through the Status Register, Mask Register, Chaining Register, and Software Request register.

Once the channel is programmed and enabled, the DMA process may be initiated in one of two ways, either by a hardware DMA request (DREQn) or a software request (Software Request Register).

Once programmed to a particular Process/Mode configuration, the channel will operate in that configuration until programmed otherwise. For this reason, restarting a channel after the current buffer expires does not require complete reprogramming of the channel. Only those parameters which have changed need to be reprogrammed. The Byte Count

Register is always changed and must be reprogrammed. A Target or Requester Address Register which is incremented or decremented should be reprogrammed also.

3.6.1 BUFFER PROCESSES

The Buffer Process is determined by the Auto-Initialize bit of Mode Register I and the Chaining Register. If Auto-Initialize is enabled, Chaining should not be used.

3.6.1.1 Single Buffer Process

The Single Buffer Process is programmed by disabling Chaining via the Chaining Register and programming Mode Register I for non-Auto-Initialize.

3.6.1.2 Buffer Auto-Initialize Process

Setting the Auto-Initialize bit in Mode Register I is all that is necessary to place the channel in this mode. Buffer Auto-Initialize must not be enabled simultaneous to enabling the Buffer Chaining Mode as this will have unpredictable results.

Once the Base Registers are loaded, the channel is ready to be enabled. The channel will reload its Current Registers from the Base Registers each time the Current Buffer expires, either by an expired Byte Count or an external EOP#.

3.6.1.3 Buffer Chaining Process

The Buffer Chaining Process is entered into from the Single Buffer Process. The Mode Registers should be programmed first, with all of the Transfer Modes defined as if the channel were to operate in the Single Buffer Process. The channel's Base and Current Registers are then loaded. When the channel has been set up in this way, and the chaining interrupt service routine is in place, the Chaining Process can be entered by programming the Chaining Register. Figure 3.23 illustrates the Buffer Chaining Process.

An interrupt (IRQ1) will be generated immediately after the Chaining Process is entered, as the channel

then perceives the Base Registers as empty and in need of reloading. It is important to have the interrupt service routine in place at the time the Chaining Process is entered into. The interrupt request is removed when the most significant byte of the Base Target Address is loaded.

The interrupt will occur again when the first buffer expires and the Current Registers are loaded from the Base Registers. The cycle continues until the Chaining Process is disabled, or the host fails to respond to IRQ1 before the Current Buffer expires.

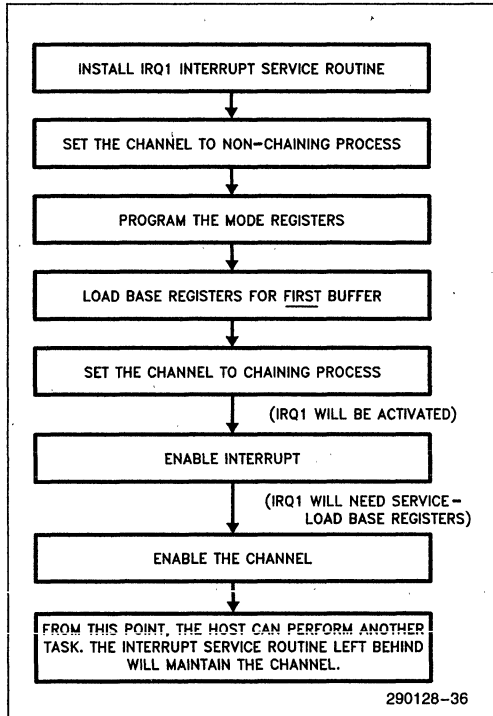


Figure 3-23. Flow of Events in the Buffer Chaining Process

Exiting the Chaining Process can be done by resetting the Chaining Mode Register. If an interrupt is pending for the channel when the Chaining Register is reset, the interrupt request will be removed. The Chaining Process can be temporarily disabled by setting the channel's Mask bit in the Mask Register.

The interrupt service routine for IRQ1 has the responsibility of reloading the Base Register as necessary. It should check the status of the channel to determine the cause of channel expiration, etc. It should also have access to operating system information regarding the channel, if any exists. The IRQ1 service routine should be capable of determining whether the chain should be continued or terminated and act on that information.

3.6.2 DATA TRANSFER MODES

The Data Transfer Modes are selected via Mode Register I. The Demand, Single, and Block Modes are selected by bits D6 and D7. The individual transfer type (Fly-By vs Two-Cycle, Read-Write-Verify, and I/O vs Memory) is programmed through both of the Mode registers.

3.6.3 CASCADED BUS MASTERS

The Cascade Mode is set by writing ones to D7 and D6 of Mode Register I. When a channel is programmed to operate in the Cascade Mode, all of the other modes associated with Mode Registers I and II are ignored. The priority and DREQn/EOP# definitions of the Command Registers will have the same effect on the channel's operation as any other mode.

3.6.4 SOFTWARE COMMANDS

There are five port addresses which, when written to, command certain operations to be performed by the 82380 DMA Controller. The data written to these locations is not of consequence, writing to the location is all that is necessary to command the 82380 to perform the indicated function. Following are descriptions of the command function.

Clear Byte Pointer Flip-Flop—location 000CH

Resets the Byte Pointer Flip-Flop. This command should be performed at the beginning of any access to the channel registers in order to be assured of beginning at a predictable place in the register programming sequence.

Master Clear—location 000DH

All DMA functions are set to their default states. This command is the equivalent of a hardware reset to the DMA Controller. Functions other than those in the DMA Controller section of the 82380 are not affected by this command.

Clear Mask Register —Channels 0–3—location 000EH
Channels 4–7—location 00CEH

This command simultaneously clears the Mask Bits of all channels in the addressed group, enabling all of the channels in the group.

Clear TC Interrupt Request—location 001EH

This command resets the Terminal Count Interrupt Request Flip-Flop. It is provided to allow the program which made a software DMA request to acknowledge that it has responded to the expiration of the requested channel(s).

3.7 Register Definitions

The following diagrams outline the bit definitions and functions of the 82380 DMA Controller's Status and Control Registers. The function and programming of the registers is covered in the previous section on DMA Controller Programming. An entry of 'X' as a bit value indicates "don't care."

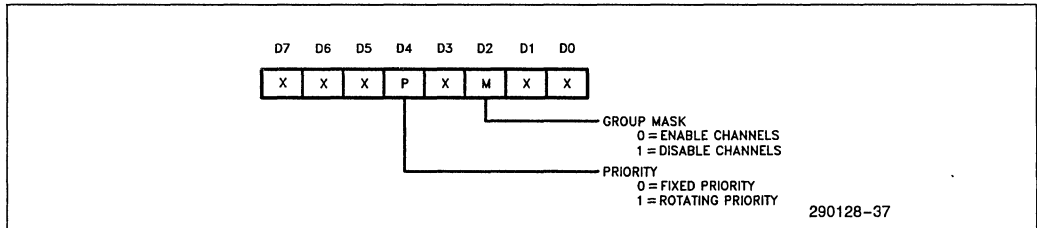
Channel Registers Channel	Register Name	(Read Current, Write Base)		Bits Accessed
		Address (Hex)	Byte Pointer	
Channel 0	Target Address	00	0	0–7
			1	8–15
		87	x	16–23
	Byte Count	10	0	24–31
		01	0	0–7
			1	8–15
	Requester Address	11	0	16–23
		90	0	0–7
			1	8–15
91		0	16–23	
		1	24–31	
Channel 1	Target Address	02	0	0–7
			1	8–15
		83	x	16–23
	Byte Count	12	0	24–31
		03	0	0–7
			1	8–15
	Requester Address	13	0	16–23
		92	0	0–7
			1	8–15
93		0	16–23	
		1	24–31	

Channel Registers Channel	Register Name	(Read Current, Write Base)		Bits Accessed
		Address (Hex)	Byte Pointer	
Channel 2	Target Address	04	0	0-7
			1	8-15
		81	x	16-23
	Byte Count	14	0	24-31
		05	0	0-7
			1	8-15
	Requester Address	15	0	16-23
		94	0	0-7
			1	8-15
95		0	16-23	
		1	24-31	
Channel 3	Target Address	06	0	0-7
			1	8-15
		82	x	16-23
	Byte Count	16	0	24-31
		07	0	0-7
			1	8-15
	Requester Address	17	0	16-23
		96	0	0-7
			1	8-15
97		0	16-23	
		1	24-31	
Channel 4	Target Address	C0	0	0-7
			1	8-15
		8F	x	16-23
	Byte Count	D0	0	24-31
		C1	0	0-7
			1	8-15
	Requester Address	D1	0	16-23
		98	0	0-7
			1	8-15
99		0	16-23	
		1	24-31	
Channel 5	Target Address	C2	0	0-7
			1	8-15
		8B	x	16-23
	Byte Count	D2	0	24-31
		C3	0	0-7
			1	8-15
	Requester Address	D3	0	16-23
		9A	0	0-7
			1	8-15
9B		0	16-23	
		1	24-31	

Channel Registers Channel	Register Name	(Read Current, Write Base) Address (Hex)	Byte Pointer	Bits Accessed		
Channel 6	Target Address	C4	0	0-7		
			1	8-15		
	Byte Count		89	x	16-23	
			D4	0	24-31	
		Requester Address	C5	0	0-7	
					1	8-15
			D5	0	16-23	
					0	0-7
	9C	0	8-15			
		1	16-23			
	9D	0	24-31			
			1	24-31		
Channel 7	Target Address	C6	0	0-7		
			1	8-15		
	Byte Count		8A	x	16-23	
			D6	0	24-31	
		Requester Address	C7	0	0-7	
					1	8-15
			D7	0	16-23	
					0	0-7
				9E	0	8-15
					1	16-23
	9F	0	24-31			
			1	24-31		

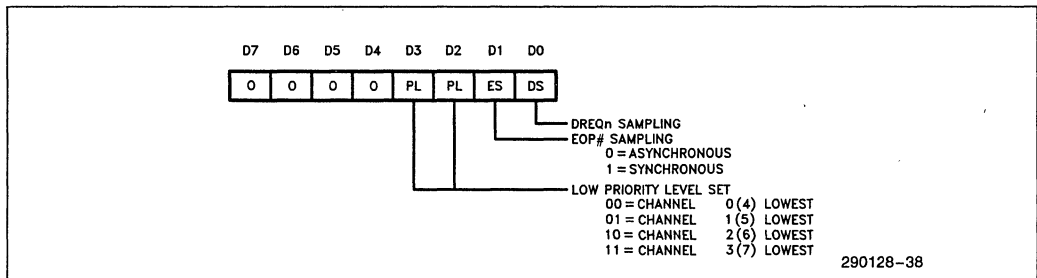
Command Register I (Write Only)

Port Address—Channels 0-3—0008H
Channels 4-7—00C8H



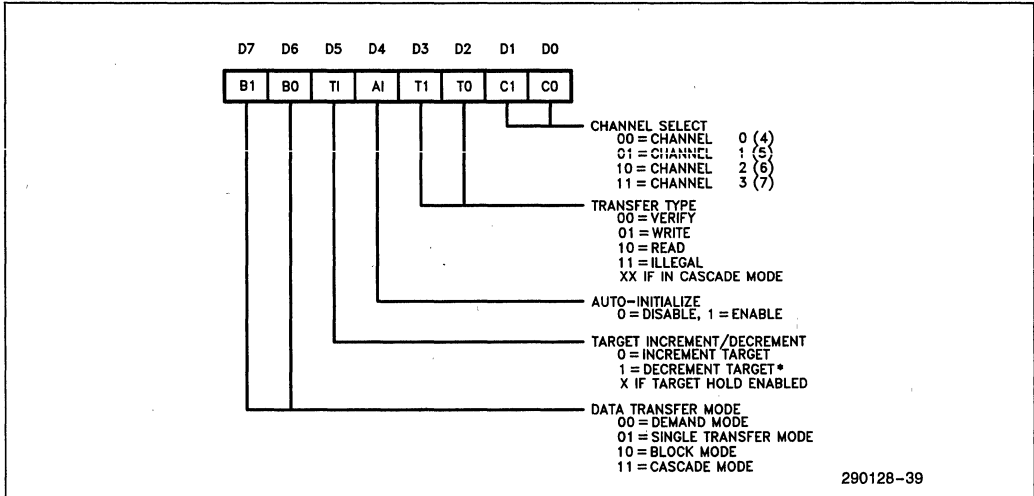
Command Register II (Write Only)

Port Addresses—Channels 0-3—001AH
Channels 4-7—00DAH



Mode Register I (Write Only)

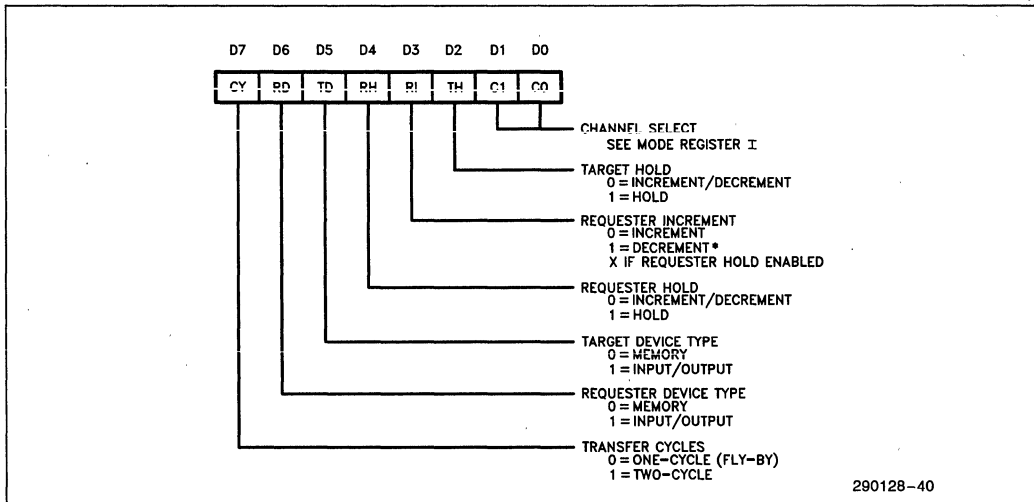
Port Addresses—Channels 0–3—000BH
Channels 4–7—00CBH



* Target and Requester DECREMENT is allowed only for byte transfers.

Mode Register II (Write Only)

Port Addresses—Channels 0–3—001BH
Channels 4–7—00DBH

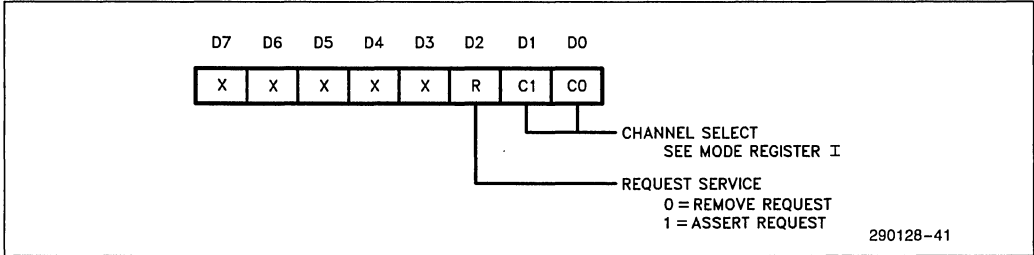


* Target and Requester DECREMENT is allowed only for byte transfers.

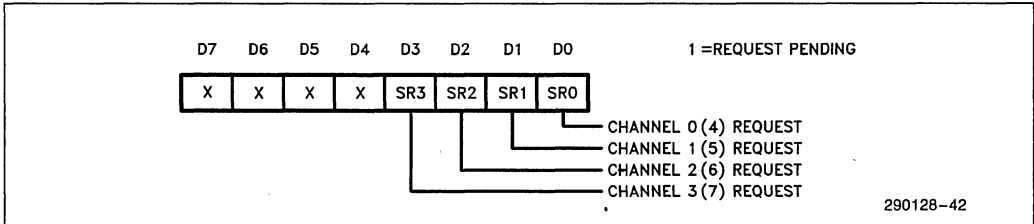
Software Request Register (Read/Write)

Port Addresses—Channels 0–3—0009H
 Channels 4–7—00C9H

Write Format: Software DMA Service Request

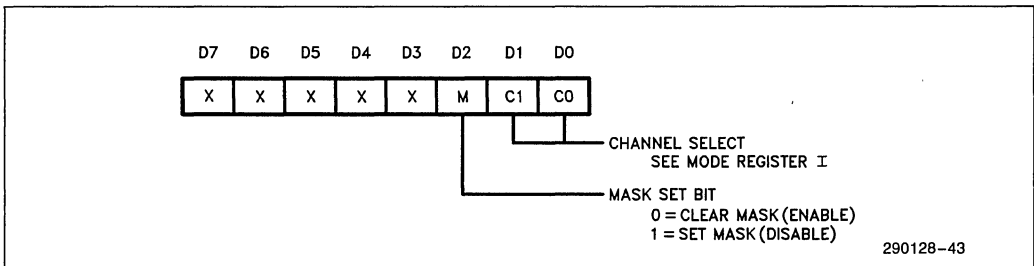


Read Format: Software Requests Pending



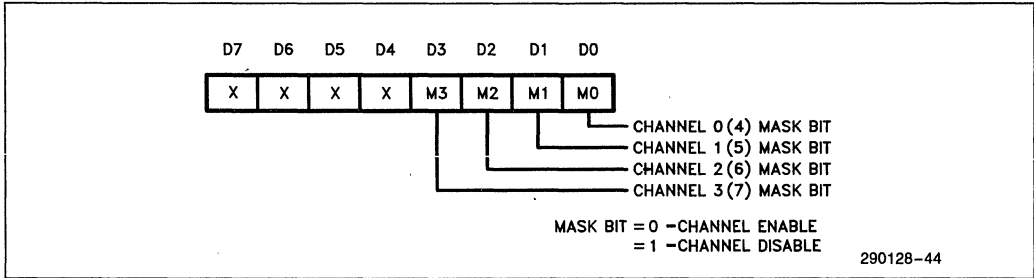
Mask Set/Reset Register Individual Channel Mask (Write Only)

Port Addresses—Channels 0–3—000AH
 Channels 4–7—00CAH



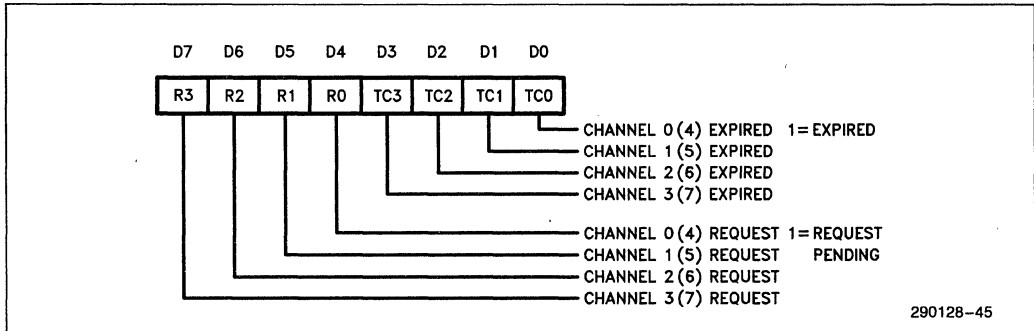
Mask Read/Write Register Group Channel Mask (Read/Write)

Port Addresses—Channels 0–3—000FH
 Channels 4–7—00CFH



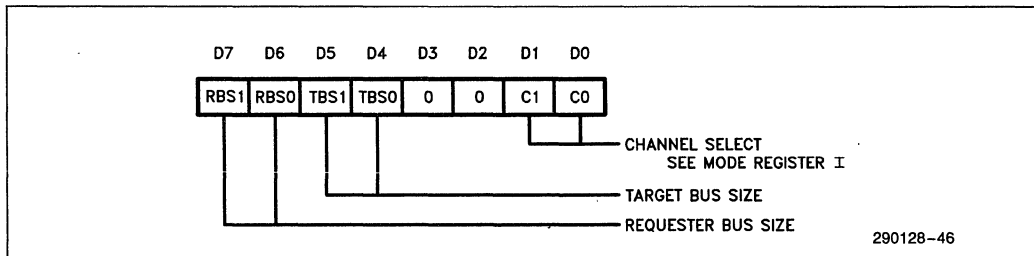
Status Register Channel Process Status (Read Only)

Port Addresses—Channels 0–3—0008H
 Channels 4–7—00C8H



Bus Size Register Set Data Path Width (Write Only)

Port Addresses—Channels 0–3—0018H
 Channels 4–7—00D8H

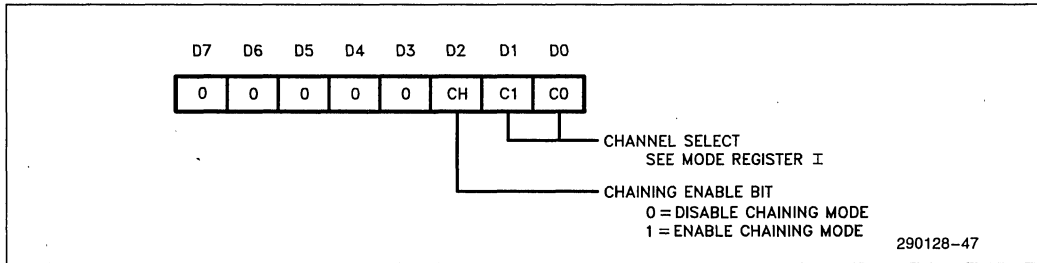


Bus Size Encoding:
 00 = Reserved by Intel 10 = 16-bit Bus
 01 = 32-bit Bus 11 = 8-bit Bus

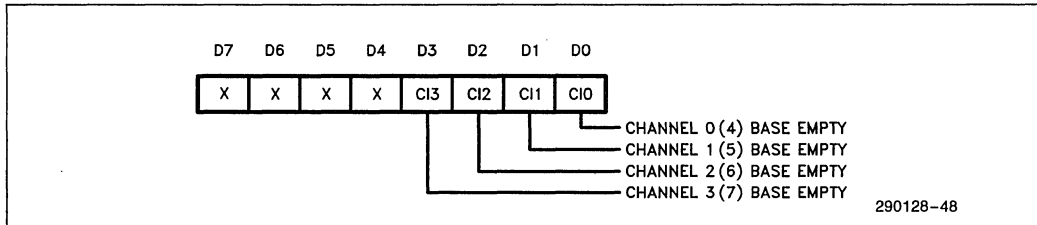
Chaining Register (Read/Write)

Port Addresses—Channels 0–3—0019H
Channels 4–7—00D9H

Write Format: Set Chaining Mode



Read Format: Channel Interrupt Status



3.8 8237A Compatibility

The register arrangement of the 82380 DMA Controller is a superset of the 8237A DMA Controller. Functionally the 82380 DMA Controller is very different from the 8237A. Most of the functions of the 8237A are performed also by the 82380. The following discussion points out the differences between the 8237A and the 82380.

The 8237A is limited to transfers between I/O and memory only (except in one special case, where two channels can be used to perform memory-to-memory transfers). The 82380 DMA Controller can transfer between any combination of memory and I/O. Several other features of the 8237A are enhanced or expanded in the 82380 and other features are added.

The 8237A is an 8-bit only DMA device. For programming compatibility, all of the 8-bit registers are preserved in the 82380. The 82380 is programmed via 8-bit registers. The address registers in the 82380 are 32-bit registers in order to support the

80386's 32-bit bus. The Byte Count Registers are 24-bit registers, allowing support of larger data blocks than possible with the 8237A.

All of the 8237A's operating modes are supported by the 82380 (except the cumbersome two-channel memory-to-memory transfer). The 82380 performs memory-to-memory transfers using only one channel. The 82380 has the added features of buffer pipelining (Buffer Chaining Process), programmable priority levels, and Byte Assembly.

The 82380 also adds the feature of address registers for both destination and source. These addresses may be incremented, decremented, or held constant, as required by the application of the individual channel. This allows any combination of destination and source device.

Each DMA channel has associated with it a Target and a Requester. In the 8237A, the Target is the device which can be accessed by the address register, the Requester is the device which is accessed by the DMA Acknowledge signals and must be an I/O device.

4.0 Programmable Interrupt Controller (PIC)

4.1 Functional Description

The 82380 Programmable Interrupt Controller (PIC) consists of three enhanced 82C59A Interrupt Controllers. These three controllers together provide 15 external and 5 internal interrupt request inputs. Each external request input can be cascaded with an additional 82C59A slave collector. This scheme allows the 82380 to support a maximum of 120 (15 x 8) external interrupt request inputs.

Following one or more interrupt requests, the 82380 PIC issues an interrupt signal to the 80386. When the 80386 host processor responds with an interrupt acknowledge signal, the PIC will arbitrate between the pending interrupt requests and place the interrupt vector associated with the highest priority pending request on the data bus.

The major enhancement in the 82380 PIC over the 82C59A is that each of the interrupt request inputs

can be individually programmed with its own interrupt vector, allowing more flexibility in interrupt vector mapping.

4.1.1 INTERNAL BLOCK DIAGRAM

The block diagram of the 82380 Programmable Interrupt Controller is shown in Figure 4-1. Internally, the PIC consists of three 82C59A banks: A, B and C. The three banks are cascaded to one another: C is cascaded to B, B is cascaded to A. The INT output of Bank A is used externally to interrupt the 80386.

Bank A has nine interrupt request inputs (two are unused), and Banks B and C have eight interrupt request inputs. Of the fifteen external interrupt request inputs, two are shared by other functions. Specifically, the Interrupt Request 3 input (IRQ3#) can be used as the Timer 2 output (TOUT2#). This pin can be used in three different ways: IRQ3# input only, TOUT2# output only, or using TOUT2# to generate an IRQ3# interrupt request. Also, the Interrupt Request 9 input (IRQ9#) can be used as DMA Request 4 input (DREQ4). Typically, only IRQ9# or DREQ4 can be used at a time.

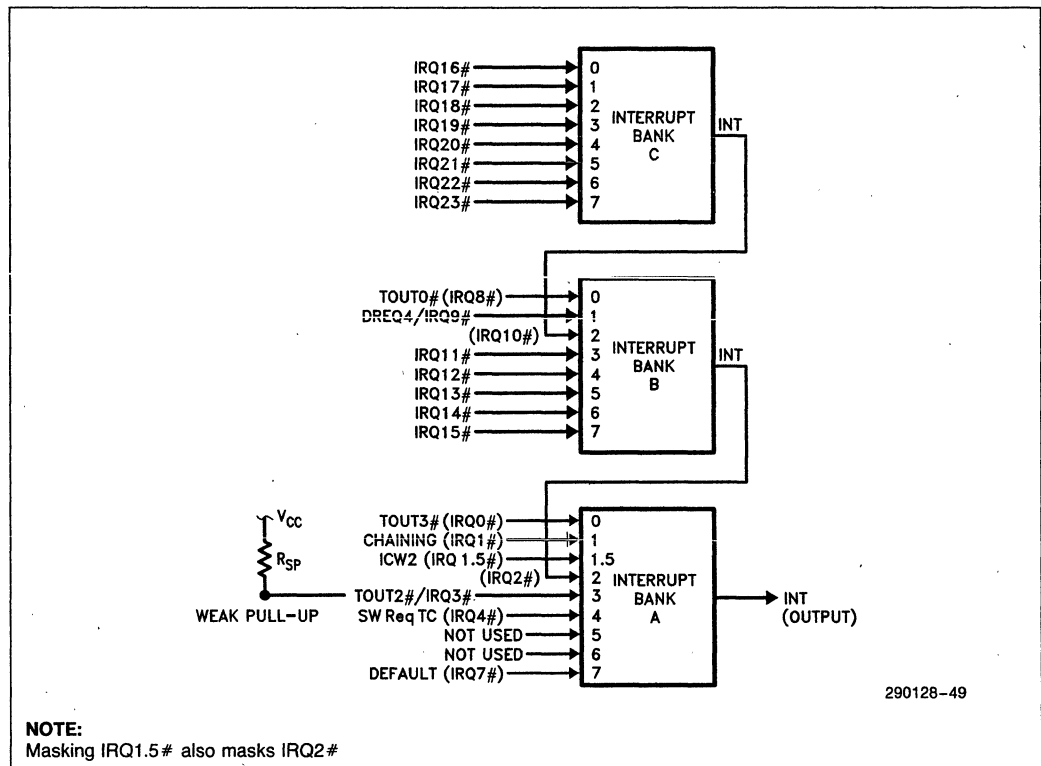


Figure 4-1. Interrupt Controller Block Diagram

4.1.2 INTERRUPT CONTROLLER BANKS

All three banks are identical, with the exception of the IRQ1.5 on Bank A. Therefore, only one bank will be discussed. In the 82380 PIC, all external requests can be cascaded into and each interrupt controller bank behaves like a master. As compared to the 82C59A, the enhancements in the banks are:

- All interrupt vectors are individually programmable. (In the 82C59A, the vectors must be programmed in eight consecutive interrupt vector locations.)

- The cascade address is provided on the Data Bus (D0–D7). (In the 82C59A, three dedicated control signals (CAS0, CAS1, CAS2) are used for master/slave cascading.)

The block diagram of a bank is shown in Figure 4-2. As can be seen from this figure, the bank consists of six major blocks: the Interrupt Request Register (IRR), the In-Service Register (ISR), the Interrupt Mask Register (IMR), the Priority Resolver (PR), the Vector Register (VR), and the Control Logic. The functional description of each block follows.

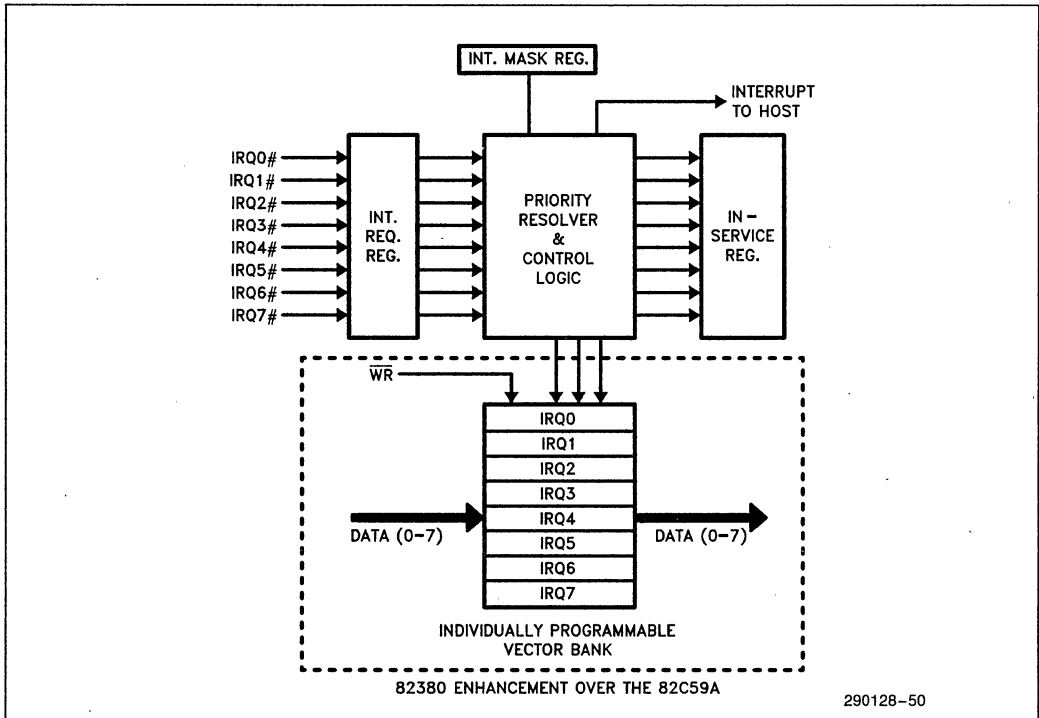


Figure 4-2. Interrupt Bank Block Diagram

INTERRUPT REQUEST (IRR) AND IN-SERVICE REGISTER (ISR)

The interrupts at the Interrupt Request (IRQ) input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service Register (ISR). The IRR is used to store all interrupt levels which are requesting service; and the ISR is used to store all interrupt levels which are being serviced.

PRIORITY RESOLVER (PR)

This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during an Interrupt Acknowledge cycle.

INTERRUPT MASK REGISTER (IMR)

The IMR stores the bits which mask the interrupt lines to be masked (disabled). The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower priority.

VECTOR REGISTERS (VR)

This block contains a set of Vector Registers, one for each interrupt request line, to store the pre-programmed interrupt vector number. The corresponding vector number will be driven onto the Data Bus of the 82380 during the Interrupt Acknowledge cycle.

CONTROL LOGIC

The Control Logic coordinates the overall operations of the other internal blocks within the same bank. This logic will drive the Interrupt Output signal (INT) HIGH when one or more unmasked interrupt inputs are active (LOW). The INT output signal goes directly to the 80386 (in Bank A) or to another bank to which this bank is cascaded (see Figure 4-1). Also, this logic will recognize an Interrupt Acknowledge cycle (via M/IO#, D/C# and W/R# signals). During this bus cycle, the Control Logic will enable the corresponding Vector Register to drive the interrupt vector onto the Data Bus.

In Bank A, the Control Logic is also responsible for handling the special ICW2 interrupt request input (IRQ1.5#).

4.2 Interface Signals

4.2.1 INTERRUPT INPUTS

There are 15 external Interrupt Request inputs and 5 internal Interrupt Requests. The external request inputs are: IRQ3#, IRQ9#, IRQ11# to IRQ23#. They are shown in bold arrows in Figure 4-1. All IRQ inputs are active LOW and they can be programmed (via a control bit in the Initialization Command Word 1 (ICW1)) to be either edge-triggered or level-triggered. In order to be recognized as a valid interrupt request, the interrupt input must be active (LOW) until the first INTA# cycle (see Bus Functional Description).

Note that all 15 external Interrupt Request inputs have weak internal pull-up resistors.

As mentioned earlier, an 82C59A can be cascaded to each external interrupt input to expand the interrupt capacity to a maximum of 120 levels. Also, two of the interrupt inputs are dual functions: IRQ3# can be used as Timer 2 output (TOUT2#) and IRQ9# can be used as DREQ4 input. IRQ3# is a bidirectional dual function pin. This interrupt request input is wired-OR with the output of Timer 2 (TOUT2#). If only IRQ3# function is to be used, Timer 2 should be programmed so that OUT2 is LOW. Note that TOUT2# can also be used to generate an interrupt request to IRQ3# input.

The five internal interrupt requests serve special system functions. They are shown in Table 4-1. The following paragraphs describe these interrupts.

Table 4-1. 82380 Internal Interrupt Requests

Interrupt Request	Interrupt Source
IRQ0#	Timer 3 Output (TOUT3#)
IRQ8#	Timer 0 Output (TOUT0#)
IRQ1#	DMA Chaining Request
IRQ4#	DMA Terminal Count
IRQ1.5#	ICW2 Written

TIMER 0 AND TIMER 3 INTERRUPT REQUESTS [IRQ#]

IRQ8# and IRQ0# interrupt requests are initiated by the output of Timers 0 and 3, respectively. Each of these requests is generated by an edge-detector flip-flop. The flip-flops are activated by the following conditions:

- Set— Rising edge of timer output (TOUT);
- Clear— Interrupt acknowledge for this request; OR Request is masked (disabled); OR Hardware Reset.

CHAINING AND TERMINAL COUNT INTERRUPTS [IRQ1#]

These interrupt requests are generated by the 82380 DMA Controller. The chaining request (IRQ1#) indicates that the DMA Base Register is not loaded. The Terminal Count request (IRQ4#) indicates that a software DMA request was cleared.

ICW2 INTERRUPT REQUEST [IRQ1.5#]

Whenever an Initialization Control Word 2 (ICW2) is written to a Bank, a special ICW2 interrupt request is generated. The interrupt will be cleared when the newly programmed ICW2 Register is read. This interrupt request is in Bank A at level 1.5. This interrupt request is internally ORed with the Cascaded Request from Bank B and is always assigned a higher priority than the Cascaded Request.

This special interrupt is provided to support compatibility with the original 82C59A. A detailed description of this interrupt is discussed in the Programming section.

DEFAULT INTERRUPT [IRQ7#]

During an Interrupt Acknowledge cycle, if there is no active pending request, the PIC will automatically

generate a default vector. This vector corresponds to the IRQ7# vector in Bank A.

4.2.2 INTERRUPT OUTPUT (INT)

The INT output pin is taken directly from bank A. This signal should be tied to the Maskable Interrupt Request (INTR) of the 80386. When this signal is active (HIGH), it indicates that one or more internal/external interrupt requests are pending. The 80386 is expected to respond with an interrupt acknowledge cycle.

4.3 Bus Functional Description

The INT output of bank A will be activated as a result of any unmasked interrupt request. This may be a non-cascaded or cascaded request. After the PIC has driven the INT signal HIGH, 80386 will respond by performing two interrupt acknowledge cycles. The timing diagram in Figure 4-3 shows a typical interrupt acknowledge process between the 82380 and the 80386 CPU.

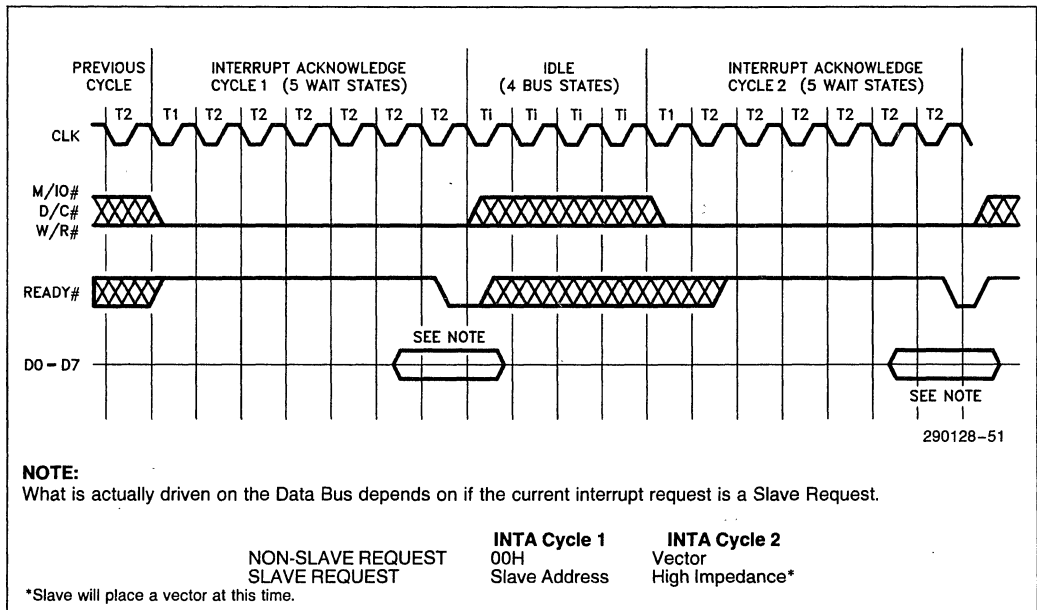


Figure 4-3. Interrupt Acknowledge Cycle

After activating the INT signal, the 82380 monitors the status lines (M/IO#, D/C#, W/R#) and waits for the 80386 to initiate the first interrupt acknowledge cycle. In the 80386 environment, two successive interrupt acknowledge cycles (INTA) marked by M/IO# = LOW, D/C# = LOW, and W/R# = LOW are performed. During the first INTA cycle, the PIC will determine the highest priority request. Assuming this interrupt input has no external Slave Controller cascaded to it, the 82380 will drive the Data Bus with 00H in the first INTA cycle. During the second INTA cycle, the 82380 PIC will drive the Data Bus with the corresponding preprogrammed interrupt vector.

If the PIC determines (from the ICW3) that this interrupt input has an external Slave Controller cascaded to it, it will drive the Data Bus with the specific Slave Cascade Address (instead of 00H) during the first INTA cycle. This Slave Cascade Address is the preprogrammed content in the corresponding Vector Register. This means that no Slave Address should be chosen to be 00H. Note that the Slave Address and Interrupt Vector are different interpretations of the same thing. They are both the contents of the programmable Vector Register. During the second INTA cycle, the Data Bus will be floated so that the external Slave Controller can drive its interrupt vector on the bus. Since the Slave Interrupt Controller resides on the system bus, bus transceiver enable and direction control logic must take this into consideration.

In order to have a successful interrupt service, the interrupt request input must be held active (LOW) until the beginning of the first interrupt acknowledge cycle. If there is no pending interrupt request when the first INTA cycle is generated, the PIC will generate a default vector, which is the IRQ7 vector (bank A level 7).

According to the Bus Cycle definition of the 80386, there will be four Bus Idle States between the two interrupt acknowledge cycles. These idle bus cycles will be initiated by the 80386. Also, during each interrupt acknowledge cycle, the internal Wait State Generator of the 82380 will automatically generate the required number of wait states for internal delays.

4.4 Mode of Operation

A variety of modes and commands are available for controlling the 82380 PIC. All of them are programmable; that is, they may be changed dynamically under software control. In fact, each bank can be programmed individually to operate in different modes. With these modes and commands, many possible

configurations are conceivable, giving the user enough versatility for almost any interrupt controlled application.

This section is not intended to show how the 82380 PIC can be programmed. Rather, it describes the operation in different modes.

4.4.1 END-OF-INTERRUPT

Upon completion of an interrupt service routine, the interrupted bank needs to be notified so its ISR can be updated. This allows the PIC to keep track of which interrupt levels are in the process of being serviced and their relative priorities. Three different End-Of-Interrupt (EOI) formats are available. They are: Non-Specific EOI Command, Specific EOI Command, and Automatic EOI Mode. Selection of which EOI to use is dependent upon the interrupt operations the user wishes to perform.

If the 82380 is NOT programmed in the Automatic EOI Mode, an EOI command must be issued by the 80386 to the specific 82380 PIC Controller Bank. Also, if this controller bank is cascaded to another internal bank, an EOI command must also be sent to the bank to which this bank is cascaded. For example, if an interrupt request of Bank C in the 82380 PIC is serviced, an EOI should be written into Bank C, Bank B and Bank A. If the request comes from an external interrupt controller cascaded to Bank C, then an EOI should be written into the external controller as well.

NON-SPECIFIC EOI COMMAND

A Non-Specific EOI command sent from the 80386 lets the 82380 PIC bank know when a service routine has been completed, without specification of its exact interrupt level. The respective interrupt bank automatically determines the interrupt level and resets the correct bit in the ISR.

To take advantage of the Non-Specific EOI, the interrupt bank must be in a mode of operation in which it can predetermine its in-service routine levels. For this reason, the Non-Specific EOI command should only be used when the most recent level acknowledged and serviced is always the highest priority level (i.e., in the Fully Nested Mode structure to be described below). When the interrupt bank receives a Non-Specific EOI command, it simply resets the highest priority ISR bit to indicate that the highest priority routine in service is finished.

Special consideration should be taken when deciding to use the Non-Specific EOI command. Here are two operating conditions in which it is best NOT

used since the Fully Nested Mode structure will be destroyed:

- Using the Set Priority command within an interrupt service routine.
- Using a Special Mask Mode.

These conditions are covered in more detail in their own sections, but are listed here for reference.

SPECIFIC EOI COMMAND

Unlike a Non-Specific EOI command which automatically resets the highest priority ISR bit, a Specific EOI command specifies an exact ISR bit to be reset. Any one of the IRQ levels of an interrupt bank can be specified in the command.

The Specific EOI command is needed to reset the ISR bit of a completed service routine whenever the interrupt bank is not able to automatically determine it. The Specific EOI command can be used in all conditions of operation, including those that prohibit Non-Specific EOI command usage mentioned above.

AUTOMATIC EOI MODE

When programmed in the Automatic EOI Mode, the 80386 no longer needs to issue a command to notify the interrupt bank it has completed an interrupt routine. The interrupt bank accomplishes this by performing a Non-Specific EOI automatically at the end of the second INTA cycle.

Special consideration should be taken when deciding to use the Automatic EOI Mode because it may disturb the Fully Nested Mode structure. In the Automatic EOI Mode, the ISR bit of a routine in service is reset right after it is acknowledged, thus leaving no designation in the ISR that a service routine is being executed. If any interrupt request within the same bank occurs during this time and interrupts are enabled, it will get serviced regardless of its priority.

Therefore, when using this mode, the 80386 should keep its interrupt request input disabled during execution of a service routine. By doing this, higher priority interrupt levels will be serviced only after the completion of a routine in service. This guideline restores the Fully Nested Mode structure. However, in this scheme, a routine in service cannot be interrupted since the host's interrupt request input is disabled.

4.4.2 INTERRUPT PRIORITIES

The 82380 PIC provides various methods for arranging the interrupt priorities of the interrupt request inputs to suit different applications. The following subsections explain these methods in detail.

4.4.2.1 Fully Nested Mode

The Fully Nested Mode of operation is a general purpose priority mode. This mode supports a multi-level interrupt structure in which all of the Interrupt Request (IRQ) inputs within one bank are arranged from highest to lowest.

Unless otherwise programmed, the Fully Nested Mode is entered by default upon initialization. At this time, IRQ0# is assigned the highest priority (priority = 0) and IRQ7# the lowest (priority = 7). This default priority can be changed, as will be explained later in the Rotating Priority Mode.

When an interrupt is acknowledged, the highest priority request is determined from the Interrupt Request Register (IRR) and its vector is placed on the bus. In addition, the corresponding bit in the In-Service Register (ISR) is set to designate the routine in service. This ISR bit will remain set until the 80386 issues an End Of Interrupt (EOI) command immediately before returning from the service routine; or alternately, if the Automatic End Of Interrupt (AEOI) bit is set, the ISR bit will be reset at the end of the second INTA cycle.

While the ISR bit is set, all further interrupts of the same or lower priority are inhibited. Higher level interrupts can still generate an interrupt, which will be acknowledged only if the 80386 internal interrupt enable flip-flop has been re-enabled (through software inside the current service routine).

4.4.2.2 Automatic Rotation—Equal Priority Devices

Automatic rotation of priorities serves in applications where the interrupting devices are of equal priority within an interrupt bank. In this kind of environment, once a device is serviced, all other equal priority peripherals should be given a chance to be serviced before the original device is serviced again. This is accomplished by automatically assigning a device the lowest priority after being serviced. Thus, in the worst case, the device would have to wait until all other peripherals connected to the same bank are serviced before it is serviced again.

There are two methods of accomplishing automatic rotation. One is used in conjunction with the Non-Specific EOI command and the other is used with

the Automatic EOI mode. These two methods are discussed below.

ROTATE ON NON-SPECIFIC EOI COMMAND

When the Rotate On Non-Specific EOI command is issued, the highest ISR bit is reset as in a normal Non-Specific EOI command. However, after it is reset, the corresponding Interrupt Request (IRQ) level is assigned the lowest priority. Other IRQ priorities rotate to conform to the Fully Nested Mode based on the newly assigned low priority.

Figure 4-4 shows how the Rotate On Non-Specific EOI command affects the interrupt priorities. Assume the IRQ priorities were assigned with IRQ0 the highest and IRQ7 the lowest. IRQ6 and IRQ4 are already in service but neither is completed. Being the higher priority routine, IRQ4 is necessarily the routine being executed. During the IRQ4 routine, a rotate on Non-Specific EOI command is executed. When this happens, Bit 4 in the ISR is reset. IRQ4 then becomes the lowest priority and IRQ5 becomes the highest.

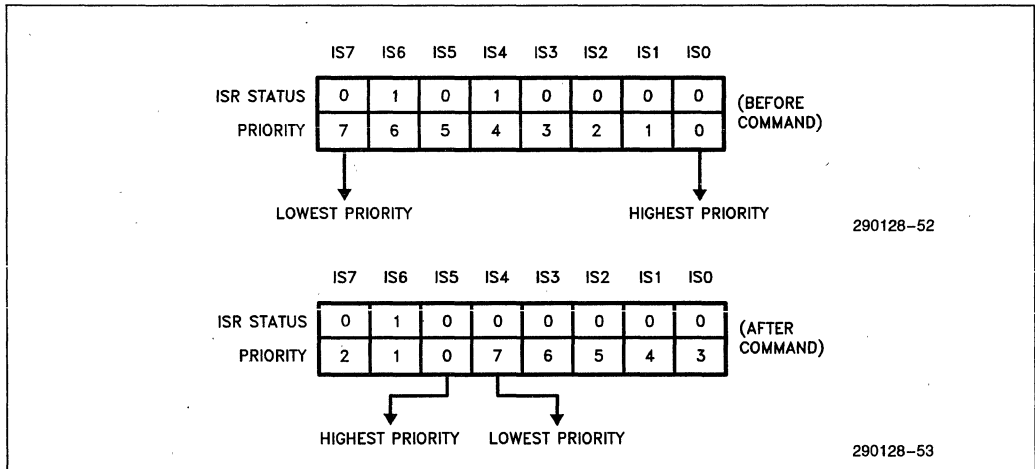


Figure 4-4. Rotate On Non-Specific EOI Command

ROTATE ON AUTOMATIC EOI MODE

The Rotate On Automatic EOI Mode works much like the Rotate On Non-Specific EOI Command. The main difference is that priority rotation is done automatically after the second INTA cycle of an interrupt request. To enter or exit this mode, a Rotate-On-Automatic-EOI Set Command and Rotate-On-Automatic-EOI Clear Command is provided. After this mode is entered, no other commands are needed as in the normal Automatic EOI Mode. However, it must be noted again that when using any form of the Automatic EOI Mode, special consideration should be taken. The guideline presented in the Automatic EOI Mode also applies here.

4.4.2.3 Specific Rotation—Specific Priority

Specific rotation gives the user versatile capabilities in interrupt controlled operations. It serves in those applications in which a specific device's interrupt priority must be altered. As opposed to Automatic Rotation which will automatically set priorities after each interrupt request is serviced, specific rotation is completely user controlled. That is, the user selects which interrupt level is to receive the lowest or the highest priority. This can be done during the main

program or within interrupt routines. Two specific rotation commands are available to the user: Set Priority Command and Rotate On Specific EOI Command.

SET PRIORITY COMMAND

The Set Priority Command allows the programmer to assign an IRQ level the lowest priority. All other interrupt levels will conform to the Fully Nested Mode based on the newly assigned low priority.

ROTATE ON SPECIFIC EOI COMMAND

The Rotate On Specific EOI Command is literally a combination of the Set Priority Command and the Specific EOI Command. Like the Set Priority Command, a specified IRQ level is assigned lowest priority. Like the Specific EOI Command, a specified level will be reset in the ISR. Thus, this command accomplishes both tasks in one single command.

4.4.2.4 Interrupt Priority Mode Summary

In order to simplify understanding the many modes of interrupt priority, Table 4-2 is provided to bring out their summary of operations.

Table 4-2. Interrupt Priority Mode Summary

Interrupt Priority Mode	Operation Summary	Effect On Priority After EOI	
		Non-Specific/Automatic	Specific
Fully-Nested Mode	IRQ0# -Highest Priority IRQ7# -Lowest Priority	No change in priority. Highest ISR bit is reset.	Not Applicable.
Automatic Rotation (Equal Priority Devices)	Interrupt level just serviced is the lowest priority. Other priorities rotate to conform to Fully-Nested Mode.	Highest ISR bit is reset and the corresponding level becomes the lowest priority.	Not Applicable.
Specific Rotation (Specific Priority Devices)	User specifies the lowest priority level. Other priorities rotate to conform to Fully-Nested Mode.	Not Applicable.	As described under 'Operation Summary'.

4.4.3 INTERRUPT MASKING

VIA INTERRUPT MASK REGISTER

Each bank in the 82380 PIC has an Interrupt Mask Register (IMR) which enhances interrupt control capabilities. This IMR allows individual IRQ masking. When an IRQ is masked, its interrupt request is disabled until it is unmasked. Each bit in the 8-bit IMR disables one interrupt channel if it is set (HIGH). Bit 0 masks IRQ0, Bit 1 masks IRQ1 and so forth. Masking an IRQ channel will only disable the corresponding channel and does not affect the others operations.

The IMR acts only on the output of the IRR. That is, if an interrupt occurs while its IMR bit is set, this request is not 'forgotten'. Even with an IRQ input masked, it is still possible to set the IRR. Therefore, when the IMR bit is reset, an interrupt request to the 80386 will then be generated, providing that the IRQ request remains active. If the IRQ request is removed before the IMR is reset, the Default Interrupt Vector (Bank A, level 7) will be generated during the interrupt acknowledge cycle.

SPECIAL MASK MODE

In the Fully Nested Mode, all IRQ levels of lower priority than the routine in service are inhibited. However, in some applications, it may be desirable to let a lower priority interrupt request to interrupt the routine in service. One method to achieve this is by using the Special Mask Mode. Working in conjunction with the IMR, the Special Mask Mode enables interrupts from all levels except the level in service. This is usually done inside an interrupt service routine by masking the level that is in service and then issuing the Special Mask Mode Command. Once the Special Mask Mode is enabled, it remains in effect until it is disabled.

4.4.4 EDGE OR LEVEL INTERRUPT TRIGGERING

Each bank in the 82380 PIC can be programmed independently for either edge or level sensing for the interrupt request signals. Recall that all IRQ inputs are active LOW. Therefore, in the edge triggered mode, an active edge is defined as an input transition from an inactive (HIGH) to active (LOW) state. The interrupt input may remain active without generating another interrupt. During level triggered mode, an interrupt request will be recognized by an active (LOW) input, and there is no need for edge detection. However, the interrupt request must be removed before the EOI Command is issued, or the 80386 must be disabled to prevent a second false interrupt from occurring.

In either modes, the interrupt request input must be active (LOW) during the first INTA cycle in order to be recognized. Otherwise, the Default Interrupt Vector will be generated at level 7 of Bank A.

4.4.5 INTERRUPT CASCADING

As mentioned previously, the 82380 allows for external Slave interrupt controllers to be cascaded to any of its external interrupt request pins. The 82380 PIC indicates that a external Slave Controller is to be serviced by putting the contents of the Vector Register associated with the particular request on the 80386 Data Bus during the first INTA cycle (instead of 00H during a non-slave service). The external logic should latch the vector on the Data Bus using the INTA status signals and use it to select the external Slave Controller to be serviced (see Figure 4-5). The selected Slave will then respond to the second INTA cycle and place its vector on the Data Bus. This method requires that if external Slave Controllers

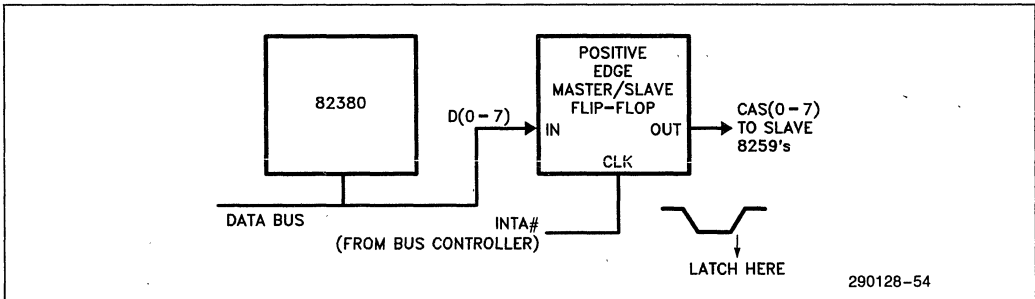


Figure 4-5. Slave Cascade Address Capturing

are used in the system, no vector should be programmed to 00H.

Since the external Slave Cascade Address is provided on the Data Bus during INTA cycle 1, an external latch is required to capture this address for the Slave Controller. A simple scheme is depicted in Figure 4-5.

4.4.5.1 Special Fully Nested Mode

This mode will be used where cascading is employed and the priority is to be conserved within each Slave Controller. The Special Fully Nested Mode is similar to the 'regular' Fully Nested Mode with the following exceptions:

- When an interrupt request from a Slave Controller is in service, this Slave Controller is not locked out from the Master's priority logic. Further interrupt requests from the higher priority logic within the Slave Controller will be recognized by the 82380 PIC and will initiate interrupts to the 80386. In comparing to the 'regular' Fully Nested Mode, the Slave Controller is masked out when its request is in service and no higher requests from the same Slave Controller can be serviced.
- Before exiting the interrupt service routine, the software has to check whether the interrupt serviced was the only request from the Slave Controller. This is done by sending a Non-Specific EOI Command to the Slave Controller and then reading its In Service Register. If there are no requests in the Slave Controller, a Non-Specific EOI can be sent to the corresponding 82380 PIC bank also. Otherwise, no EOI should be sent.

4.4.6 READING INTERRUPT STATUS

The 82380 PIC provides several ways to read different status of each interrupt bank for more flexible interrupt control operations. These include polling the highest priority pending interrupt request and reading the contents of different interrupt status registers.

4.4.6.1 Poll Command

The 82380 PIC supports status polling operations with the Poll Command. In a Poll Command, the

pending interrupt request with the highest priority can be determined. To use this command, the INT output is not used, or the 80386 interrupt is disabled. Service to devices is achieved by software using the Poll Command.

This mode is useful if there is a routine command common to several levels so that the INTA sequence is not needed. Another application is to use the Poll Command to expand the number of priority levels.

Notice that the ICW2 mechanism is not supported for the Poll Command. However, if the Poll Command is used, the programmable Vector Registers are of no concern since no INTA cycle will be generated.

4.4.6.2 Reading Interrupt Registers

The contents of each interrupt register (IRR, ISR, and IMR) can be read to update the user's program on the present status of the 82380 PIC. This can be a versatile tool in the decision making process of a service routine, giving the user more control over interrupt operations.

The reading of the IRR and ISR contents can be performed via the Operation Control Word 3 by using a Read Status Register Command and the content of IMR can be read via a simple read operation of the register itself.

4.5 Register Set Overview

Each bank of the 82380 PIC consists of a set of 8-bit registers to control its operations. The address map of all the registers is shown in Table 4-3. Since all three register sets are identical in functions, only one set will be described.

Functionally, each register set can be divided into five groups. They are: the four Initialization Command Words (ICW's), the three Operation Control Words (OCW's), the Poll/Interrupt Request/In-Service Register, the Interrupt Mask Register, and the Vector Registers. A description of each group follows.

Table 4-3. Interrupt Controller Register Address Map

Port Address	Access	Register Description
20H	Write Read	Bank B ICW1, OCW2, or OCW3 Bank B Poll, Request or In-Service Status Register
21H	Write Read	Bank B ICW2, ICW3, ICW4, OCW1 Bank B Mask Register
22H	Read	Bank B ICW2
28H	Read/Write	IRQ8 Vector Register
29H	Read/Write	IRQ9 Vector Register
2AH	Read/Write	Reserved
2BH	Read/Write	IRQ11 Vector Register
2CH	Read/Write	IRQ12 Vector Register
2DH	Read/Write	IRQ13 Vector Register
2EH	Read/Write	IRQ14 Vector Register
2FH	Read/Write	IRQ15 Vector Register
A0H	Write Read	Bank C ICW1, OCW2, or OCW3 Bank C Poll, Request or In-Service Status Register
A1H	Write Read	Bank C ICW2, ICW3, ICW4, OCW1 Bank C Mask Register
A2H	Read	Bank C ICW2
A8H	Read/Write	IRQ16 Vector Register
A9H	Read/Write	IRQ17 Vector Register
AAH	Read/Write	IRQ18 Vector Register
ABH	Read/Write	IRQ19 Vector Register
ACH	Read/Write	IRQ20 Vector Register
ADH	Read/Write	IRQ21 Vector Register
AEH	Read/Write	IRQ22 Vector Register
AFH	Read/Write	IRQ23 Vector Register
30H	Write Read	Bank A ICW1, OCW2, or OCW3 Bank A Poll, Request or In-Service Status Register
31H	Write Read	Bank A ICW2, ICW3, ICW4, OCW1 Bank A Mask Register
32H	Read	Bank ICW2
38H	Read/Write	IRQ0 Vector Register
39H	Read/Write	IRQ1 Vector Register
3AH	Read/Write	IRQ1.5 Vector Register
3BH	Read/Write	IRQ3 Vector Register
3CH	Read/Write	IRQ4 Vector Register
3DH	Read/Write	Reserved
3EH	Read/Write	Reserved
3FH	Read/Write	IRQ7 Vector Register

4.5.1 INITIALIZATION COMMAND WORDS (ICW)

Before normal operation can begin, the 82380 PIC must be brought to a known state. There are four 8-bit Initialization Command Words in each interrupt bank to setup the necessary conditions and modes for proper operation. Except for the second common word (ICW2) which is a read/write register, the other three are write-only registers. Without going into detail of the bit definitions of the command words, the following subsections give a brief description of what functions each command word controls.

ICW1

The ICW1 has three major functions. They are:

- To select between the two IRQ input triggering modes (edge-or level-triggered);
- To designate whether or not the interrupt bank is to be used alone or in the cascade mode. If the cascade mode is desired, the interrupt bank will accept ICW3 for further cascade mode programming. Otherwise, no ICW3 will be accepted;
- To determine whether or not ICW4 will be issued; that is, if any of the ICW4 operations are to be used.

ICW2

ICW2 is provided for compatibility with the 82C59A only. Its contents do not affect the operation of the interrupt bank in any way. Whenever the ICW2 of any of the three banks is written into, an interrupt is generated from Bank A at level 1.5. The interrupt request will be cleared after the ICW2 register has been read by the 80386. The user is expected to program the corresponding vector register or to use it as an indicator that an attempt was made to alter the contents. Note that each ICW2 register has different addresses for read and write operations.

ICW3

The interrupt bank will only accept an ICW3 if programmed in the external cascade mode (as indicated in ICW1). ICW3 is used for specific programming within the cascade mode. The bits in ICW3 indicate which interrupt request inputs have a Slave cascaded to them. This will subsequently affect the interrupt vector generation during the interrupt acknowledge cycles as described previously.

ICW4

The ICW4 is accepted only if it was selected in ICW1. This command word register serves two functions:

- To select either the Automatic EOI mode or software EOI mode;
- To select if the Special Nested mode is to be used in conjunction with the cascade mode.

4.5.2 OPERATION CONTROL WORDS (OCW)

Once initialized by the ICW's, the interrupt banks will be operating in the Fully Nested Mode by default and they are ready to accept interrupt requests. However, the operations of each interrupt bank can be further controlled or modified by the use of OCW's. Three OCW's are available for programming various modes and commands. Note that all OCW's are 8-bit write-only registers.

The modes and operations controlled by the OCW's are:

- Fully Nested Mode;
- Rotating Priority Mode;
- Special Mask Mode;
- Poll Mode;
- EOI Commands;
- Read Status Commands.

OCW1

OCW1 is used solely for masking operations. It provides a direct link to the Interrupt Mask Register (IMR). The 80386 can write to this OCW register to enable or disable the interrupt inputs. Reading the pre-programmed mask can be done via the Interrupt Mask Register which will be discussed shortly.

OCW2

OCW2 is used to select End-Of-Interrupt, Automatic Priority Rotation, and Specific Priority Rotation operations. Associated commands and modes of these operations are selected using the different combinations of bits in OCW2.

Specifically, the OCW2 is used to:

- Designate an interrupt level (0–7) to be used to reset a specific ISR bit or to set a specific priority. This function can be enabled or disabled;
- Select which software EOI command (if any) is to be executed (i.e., Non-Specific or Specific EOI);
- Enable one of the priority rotation operations (i.e., Rotate On Non-Specific EOI, Rotate On Automatic EOI, or Rotate on Specific EOI).

OCW3

There are three main categories of operation that OCW3 controls. That are summarized as follows:

- To select and execute the Read Status Register Commands, either reading the Interrupt Request Register (IRR) or the In-Service Register (ISR);
- To issue the Poll Command. The Poll Command will override a Read Register Command if both functions are enabled simultaneously;
- To set or reset the Special Mask Mode.

4.5.3 POLL/INTERRUPT REQUEST/IN-SERVICE STATUS REGISTER

As the name implies, this 8-bit read-only register has multiple functions. Depending on the command issued in the OCW3, the content of this register reflects the result of the command executed. For a Poll Command, the register read contains the binary code of the highest priority level requesting service (if any). For a Read IRR Command, the register content will show the current pending interrupt request(s). Finally, for a Read ISR Command, this register will specify all interrupt levels which are being serviced.

4.5.4 INTERRUPT MASK REGISTER (IMR)

This is a read-only 8-bit register which, when read, will specify all interrupt levels within the same bank that are masked.

4.5.5 VECTOR REGISTER (VR)

Each interrupt request input has an 8-bit read/write programmable vector register associated with it. The registers should be programmed to contain the interrupt vector for the corresponding request. The contents of the Vector Register will be placed on the Data Bus during the INTA cycles as described previously.

4.6 Programming

Programming the 82380 PIC is accomplished by using two types of command words: ICW's and OCW's. All modes and commands explained in the previous sections are programmable using the ICW's and OCW's. The ICW's are issued from the 80386 in a sequential format and are used to setup the banks in the 82380 PIC in an initial state of operation. The OCW's are issued as needed to vary and control the 82380 PIC's operations.

Both ICW's and OCW's are sent by the 80386 to the interrupt banks via the Data Bus. Each bank distinguishes between the different ICW's and OCW's by the I/O address map, the sequence they are issued (ICW's only), and by some dedicated bits among the ICW's and OCW's.

All three interrupt banks are programmed in a similar way. Therefore, only a single bank will be described.

4.6.1 INITIALIZATION (ICW)

Before normal operation can begin, each bank must be initialized by programming a sequence of two to four bytes written into the ICW's.

Figure 4-6 shows the initialization flow for an interrupt bank. Both ICW1 and ICW2 must be issued for any form of operation. However, ICW3 and ICW4 are used only if designated in ICW1. Once initialized, if any programming changes within the ICW's are to be made, the entire ICW sequence must be reprogrammed, not just an individual ICW.

Note that although the ICW2's in the 82380 PIC do not affect the Bank's operation, they still must be programmed in order to preserve the compatibility with the 82C59A. The contents programmed are not relevant to the overall operations of the interrupt banks. Also, whenever one of the three ICW2's is programmed, an interrupt level 1.5 in Bank A will be generated. This interrupt request will be cleared upon reading of the ICW2 registers. Since the three ICW2's share the same interrupt level and the system may not know the origin of the interrupt, all three ICW2's must be read.

However, it is not necessary to provide an interrupt service routine for the ICW2 interrupt. One way to avoid this is as follows. At the beginning of the initialization of the interrupt banks, the 80386 interrupt should be disabled. After each ICW2 register write operation is performed during the initialization, the corresponding ICW2 register is read. This read operation will clear the interrupt request of the 82380. At the end of the initialization, the 80386 interrupt is re-enabled. With this method, the 80386 will not detect the ICW2 interrupt request, thus eliminating the need of an interrupt service routine.

Certain internal setup conditions occur automatically within the interrupt bank after the first ICW (ICW1) has been issued. They are:

- The edge sensitive circuit is reset, which means that following initialization, an interrupt request input must make a HIGH-to-LOW transition to generate an interrupt;
- The Interrupt Mask Register (IMR) is cleared; that is, all interrupt inputs are enabled;
- IRQ7 input of each bank is assigned priority 7 (lowest);
- Special Mask Mode is cleared and Status Read is set to IRR;
- If no ICW4 is needed, then no Automatic-EOI is selected.

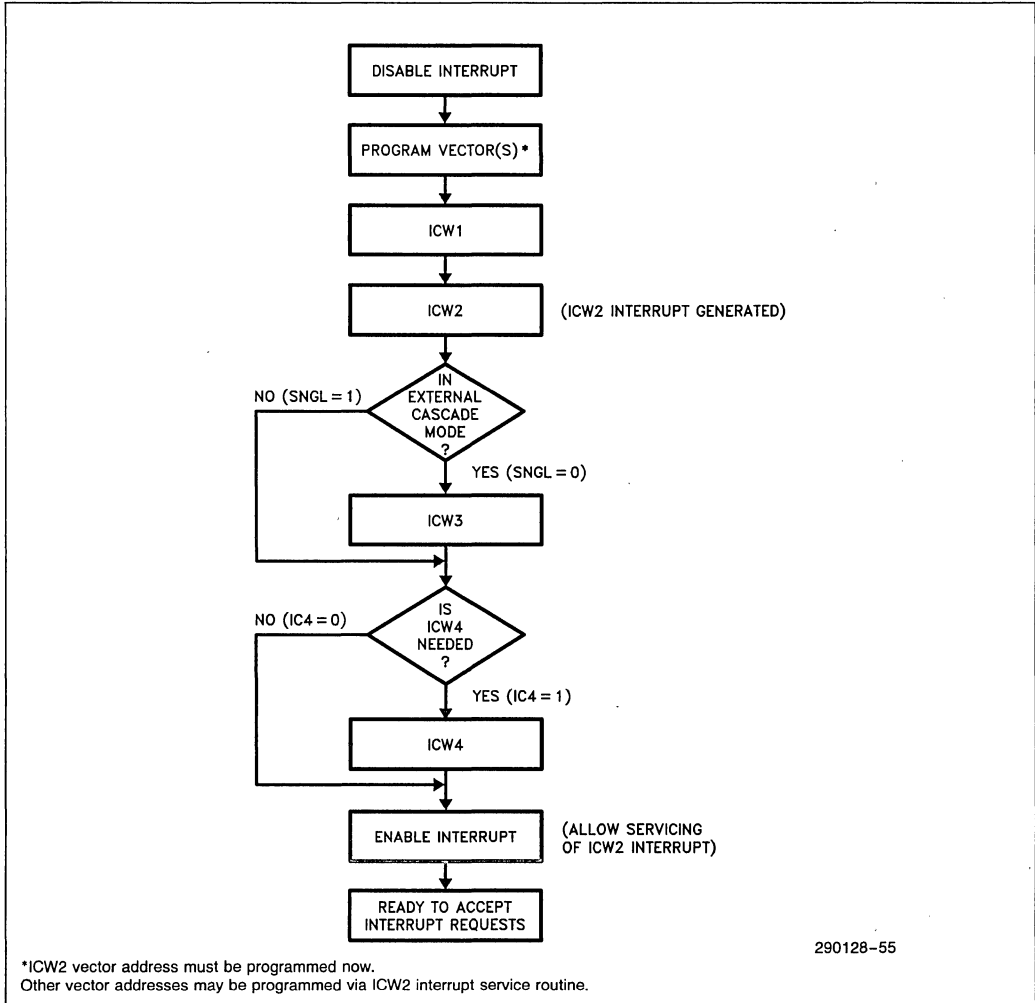


Figure 4-6. Initialization Sequence

4.6.2 VECTOR REGISTERS (VR)

Each interrupt request input has a separate Vector Register. These Vector Registers are used to store the pre-programmed vector number corresponding to their interrupt sources. In order to guarantee proper interrupt handling, all Vector Registers must be programmed with the predefined vector numbers. Since an interrupt request will be generated whenever an ICW2 is written during the initialization sequence, it is important that the Vector Register of IRQ1.5 in Bank A should be initialized and the interrupt service routine of this vector is set up before the ICW's are written.

4.6.3 OPERATION CONTROL WORDS (OCW)

After the ICW's are programmed, the operations of each interrupt controller bank can be changed by writing into the OCW's as explained before. There is no special programming sequence required for the OCW's. Any OCW may be written at any time in order to change the mode of or to perform certain operations on the interrupt banks.

4.6.3.1 Read Status and Poll Commands (OCW3)

Since the reading of IRR and ISR status as well as the result of a Poll Command are available on the

same read-only Status Register, a special Read Status/Poll Command must be issued before the Poll/Interrupt Request/In-Service Status Register is read. This command can be specified by writing the required control word into OCW3. As mentioned earlier, if both the Poll Command and the Status Read Command are enabled simultaneously, the Poll Command will override the Status Read. That is, after the command execution, the Status Register will contain the result of the Poll Command.

Note that for reading IRR and ISR, there is no need to issue a Read Status Command to the OCW3 every time the IRR or ISR is to be read. Once a Read

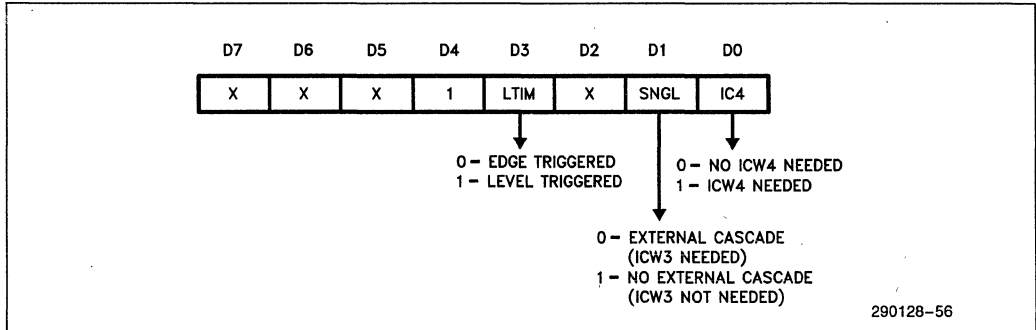
Status Command is received by the interrupt bank, it 'remembers' which register is selected. However, this is not true when the Poll Command is used.

In the Poll Command, after the OCW3 is written, the 82380 PIC treats the next read to the Status Register as an interrupt acknowledge. This will set the appropriate IS bit if there is a request and read the priority level. Interrupt Request input status remains unchanged from the Poll Command to the Status Read.

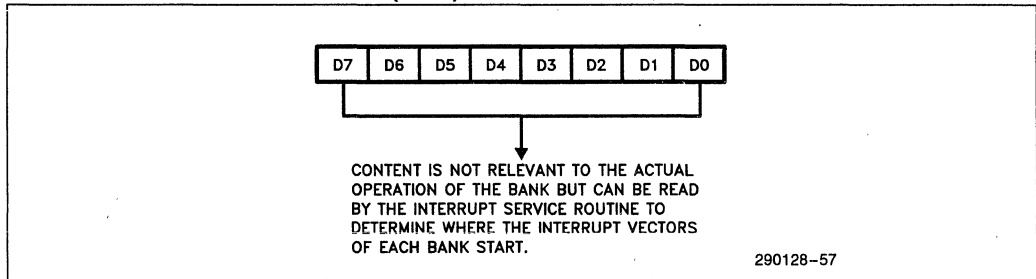
In addition to the above read commands, the Interrupt Mask Register (IMR) can also be read. When read, this register reflects the contents of the pre-programmed OCW1 which contains information on which interrupt request(s) is(are) currently disabled.

4.7 Register Bit Definition

INITIALIZATION COMMAND WORD 1 (ICW1)

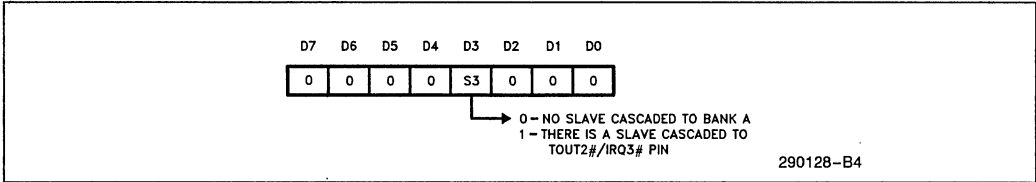


INITIALIZATION COMMAND WORD 2 (ICW2)

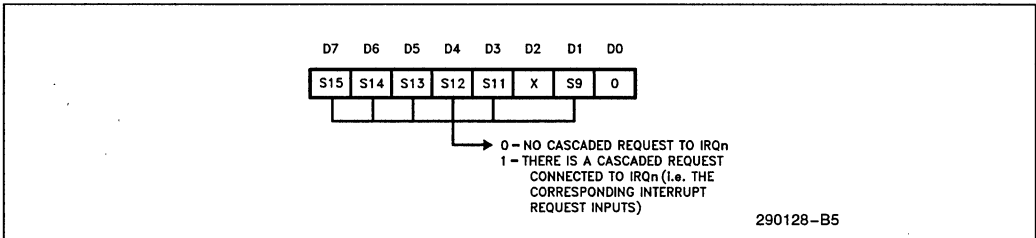


INITIALIZATION COMMAND WORD 3 (ICW3)

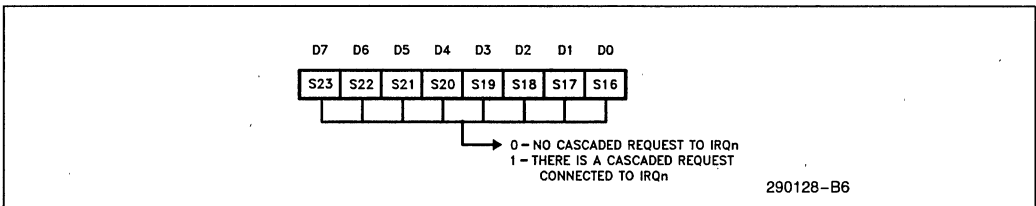
ICW3 for Bank A:



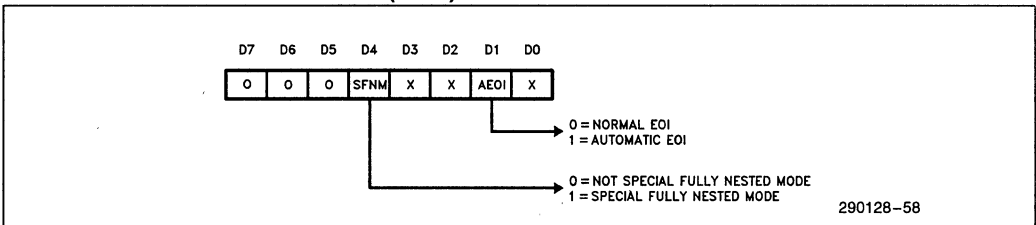
ICW3 for Bank B:



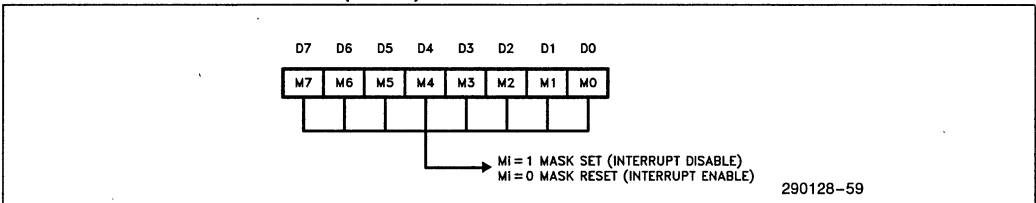
ICW3 for Bank C:



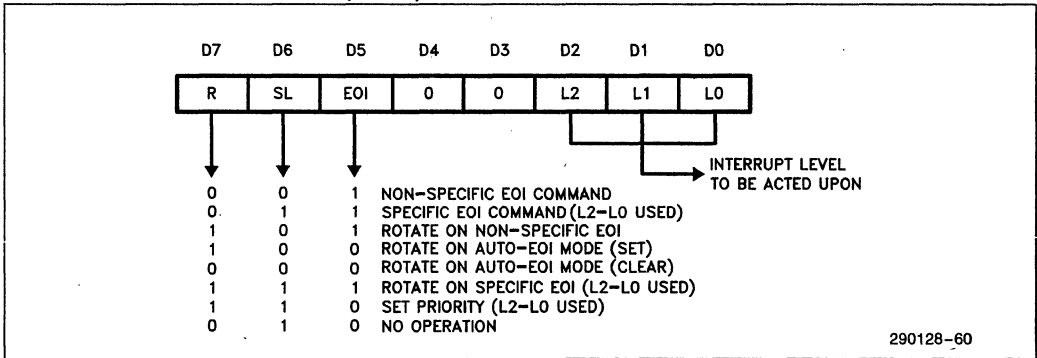
INITIALIZATION COMMAND WORD 4 (ICW4)



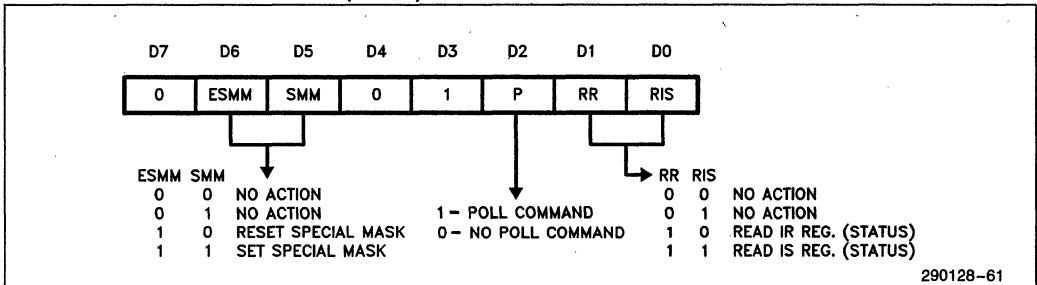
OPERATION CONTROL WORD 1 (OCW1)



OPERATION CONTROL WORD 2 (OCW2)



OPERATION CONTROL WORD 3 (OCW3)

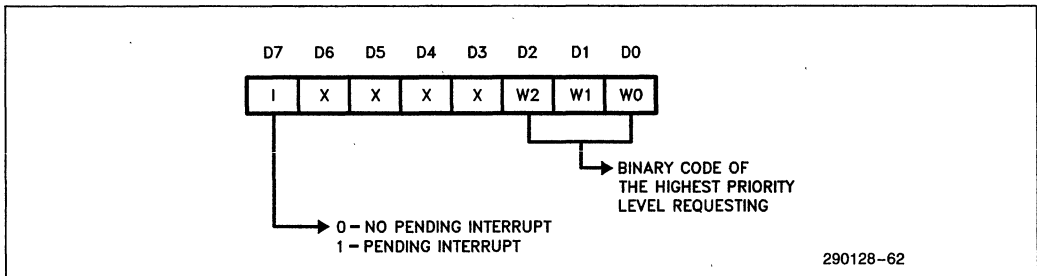


ESMM—Enable Special Mask Mode. When this bit is set to 1, it enables the SMM bit to set or reset the Special Mask Mode. When this bit is set to 0, SMM bit becomes don't care.

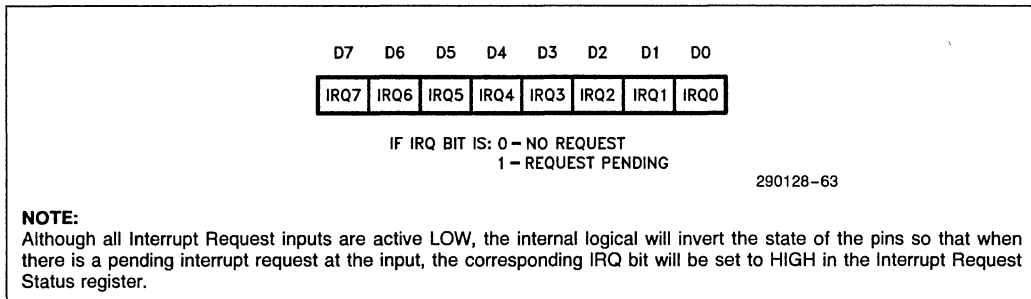
SMM—Special Mask Mode. If ESMM = 1 and SMM = 1, the interrupt controller bank will enter Special Mask Mode. If ESMM = 1 and SMM = 0, the bank will revert to normal mask mode. When ESMM = 0, SMM has no effect.

Poll/Interrupt Request/In-Service Status Register

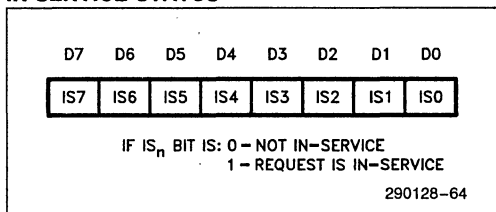
POLL COMMAND STATUS



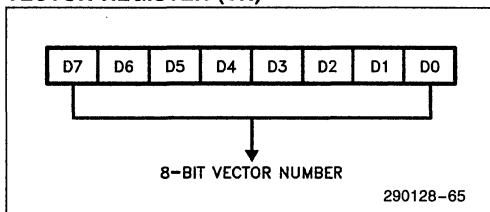
INTERRUPT REQUEST STATUS



IN-SERVICE STATUS



VECTOR REGISTER (VR)



4.8 Register Operational Summary

For ease of reference, Table 4-4 gives a summary of the different operating modes and commands with their corresponding registers.

Table 4-4 Register Operational Summary

Operational Description	Command Words	Bits
Fully Nested Mode	OCW-Default	—
Non-specific EOI Command	OCW2	EOI
Specific EOI Command	OCW2	SL, EOI, LO-L2
Automatic EOI Mode	ICW1, ICW4	IC4, AEOI
Rotate On Non-Specific EOI Command	OCW2	EOI
Rotate On Automatic EOI Mode	OCW2	R, SL, EOI
Set Priority Command	OCW2	L0-L2
Rotate On Specific EOI Command	OCW2	R, SL, EOI
Interrupt Mask Register	OCW1	M0-M7
Special Mask Mode	OCW3	ESMM, SMM
Level Triggered Mode	ICW1	LTIM
Edge Triggered Mode	ICW1	LTIM
Read Register Command, IRR	OCW3	RR, RIS
Read Register Command, ISR	OCW3	RR, RIS
Red IMR	IMR	M0-M7
Poll Command	OCW3	P
Special Fully Nested Mode	ICW2, ICW4	IC4, SFNM

5.0 PROGRAMMABLE INTERVAL TIMER

5.1 Functional Description

The 82380 contains four independently Programmable Interval Timers: Timer 0–3. All four timers are functionally compatible to the Intel 82C54. The first three timers (Timer 0–2) have specific functions. The fourth timer, Timer 3, is a general purpose timer. Table 5-1 depicts the functions of each timer. A brief description of each timer's function follows.

Table 5-1. Programmable Interval Timer Functions

Timer	Output	Function
0	IRQ8	Event Based IRQ8 Generator
1	TOUT1/REF #	Gen. Purpose/DRAM Refresh Req.
2	TOUT2# /IRQ3#	Gen. Purpose/Speaker Out/IRQ3#
3	TOUT3#	Gen. Purpose/IRQ0 Generator

TIMER 0—Event Based IRQ8 Generator

Timer 0 is intended to be used as an Event Counter. The output of this timer will generate an Interrupt Request 8 (IRQ8) upon a rising edge of the timer output (TOUT0). Typically, this timer is used to implement a time-of-day clock or system tick. The Timer 0 output is not available as an external signal.

TIMER 1—General Purpose/DRAM Refresh Request

The output of Timer 1, TOUT1, can be used as a general purpose timer or as a DRAM Refresh Request signal. The rising edge of this output creates a DRAM refresh request to the 82380 DRAM Refresh Controller. Upon reset, the Refresh Request function is disabled, and the output pin is the Timer 1 output.

TIMER 2—General Purpose/Speaker Out/IRQ3#

The Timer 2 output, TOUT2#, could be used to support tone generation to an external speaker. This pin is a bidirectional signal. When used as an input, a logic LOW asserted at this pin will generate an Interrupt Register 3 (IRQ3#) (see Programmable Interrupt Controller).

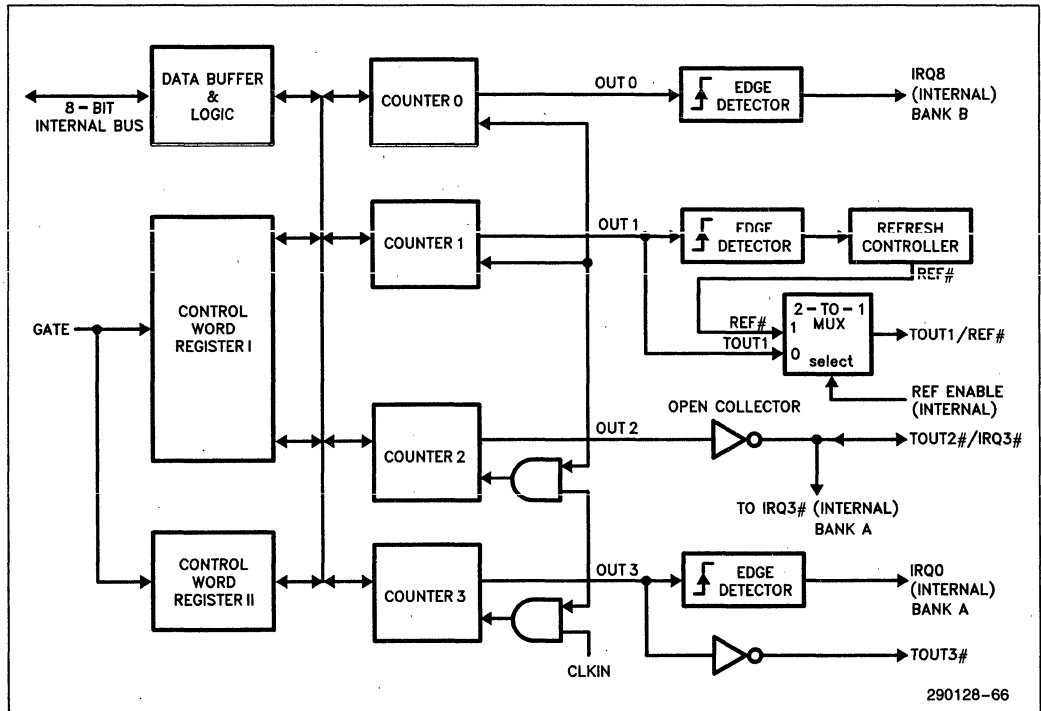


Figure 5-1. Block Diagram of Programmable Interval Timer

TIMER 3—General Purpose/Interrupt Request 0 Generator

The output of Timer 3 is fed to an edge detector and generates an Interrupt Request 0 (IRQ0) in the 82380. The inverted output of this timer (TOUT3#) is also available as an external signal for general purpose use.

5.1.1 INTERNAL ARCHITECTURE

The functional block diagram of the Programmable Interval Timer section is shown in Figure 5-1. Following is a description of each block.

DATA BUFFER & READ/WRITE LOGIC

This part of the Programmable Interval Timer is used to interface the four timers to the 82380 internal bus. The Data Buffer is for transferring commands and data between the 8-bit internal bus and the timers.

The Read/Write Logic accepts inputs from the internal bus and generates signals to control other functional blocks within the timer section.

CONTROL WORD REGISTERS I & II

The Control Word Registers are write-only registers. They are used to control the operating modes of the timers. Control Word Register I controls Timers 0, 1 and 2, and Control Word Register II controls Timer 3. Detailed description of the Control Word Registers will be included in the Register Set Overview section.

COUNTER 0, COUNTER 1, COUNTER 2, COUNTER 3

Counters 0, 1, 2, and 3 are the major parts of Timers 0, 1, 2, and 3, respectively. These four functional blocks are identical in operation, so only a single counter will be described. The internal block diagram of one counter is shown in Figure 5-2.

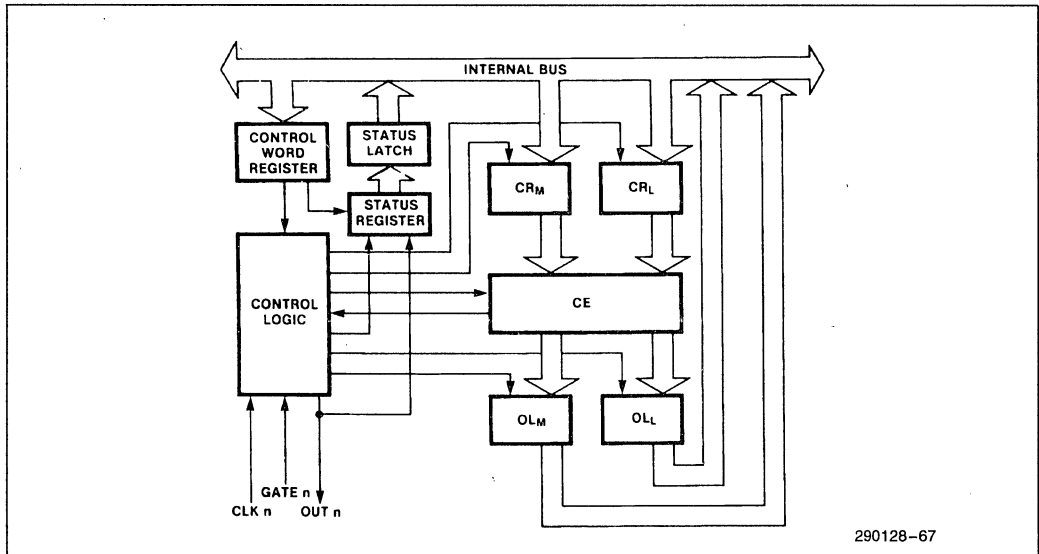


Figure 5-2. Internal Block Diagram of A Counter

The four counters share a common clock input (CLKIN), but otherwise are fully independent. Each counter is programmable to operate in a different Mode.

Although the Control Word Register is shown in the Figure 5-2, it is not part of the counter itself. Its programmed contents are used to control the operations of the counters.

The Status Register, when latched, contains the current contents of the Control Word Register and status of the output and Null Count Flag (see Read Back Command).

The Counting Element (CE) is the actual counter. It is a 16-bit presettable synchronous down counter.

The Output Latches (OL) contain two 8-bit latches (OLM and OLL). Normally, these latches 'follow' the content of the CE. OLM contains the most significant byte of the counter and OLL contains the least significant byte. If the Counter Latch Command is sent to the counter, OL will latch the present count until read by the 80386 and then return to follow the CE. One latch at a time is enabled by the timer's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that CE cannot be read. Whenever the count is read, it is one of the OL's that is being read.

When a new count is written into the counter, the value will be stored in the Count Registers (CR), and transferred to CE. The transferring of the contents from CR's to CE is defined as 'loading' of the counter. The Count Register contains two 8-bit registers: CRM (which contains the most significant byte) and CRL (which contains the least significant byte). Similar to the OL's, the Control Logic allows one register at a time to be loaded from the 8-bit internal bus. However, both bytes are transferred from the CR's to the CE simultaneously. Both CR's are cleared when the Counter is programmed. This way, if the Counter has been programmed for one byte count (either the most significant or the least significant byte only), the other byte will be zero. Note that CE cannot be written into directly. Whenever a count is written, it is the CR that is being written.

As shown in the diagram, the Control Logic consists of three signals: CLKIN, GATE, and OUT. CLKIN and GATE will be discussed in detail in the section that follows. OUT is the internal output of the counter. The external outputs of some timers (TOUT) are the inverted version of OUT (see TOUT1, TOUT2#, TOUT3#). The state of OUT depends on the mode of operation of the timer.

5.2 Interface Signals

5.2.1 CLKIN

CLKIN is an input signal used by all four timers for internal timing reference. This signal can be independent of the 82380 system clock, CLK2. In the following discussion, each 'CLK Pulse' is defined as the time period between a rising edge and a falling edge, in that order, of CLKIN.

During the rising edge of CLKIN, the state of GATE is sampled. All new counts are loaded and counters are decremented on the falling edge of CLKIN.

Please note that there are restrictions on the CLKIN signal during WRITE cycles to the 82380 timer unit. Refer to the appendix of this data manual for details on this issue.

5.2.2 TOUT1, TOUT2#, TOUT3#

TOUT1, TOUT2# and TOUT3# are the external output signals of Timer 1, Timer 2 and Timer 3, respectively. TOUT2# and TOUT3# are the inverted signals of their respective counter outputs, OUT. There is no external output for Timer 0.

If Timer 2 is to be used as a tone generator of a speaker, external buffering must be used to provide sufficient drive capability.

The Outputs of Timer 2 and 3 are dual function pins. The output pin of Timer 2 (TOUT2#/IRQ3#), which is a bidirectional open-collector signal, can also be used as interrupt request input. When the interrupt function is enabled (through the Programmable Interrupt Controller), a LOW on this input will generate an Interrupt Request 3# to the 82380 Programmable Interrupt Controller. This pin has a weak internal pull-up resistor. To use the IRQ3# function, Timer 2 should be programmed so that OUT2 is LOW. Additionally, OUT3 of Timer 3 is connected to an edge detector which will generate an Interrupt Request 0 (IRQ0) to the 82380 after the rising edge of OUT3 (see Figure 5-1).

5.2.3 GATE

GATE is not an externally controllable signal. Rather, it can be software controlled with the Internal Control Port. The state of GATE is always sampled on the rising edge of CLKIN. Depending on the mode of operation, GATE is used to enable/disable counting or trigger the start of an operation.

For Timer 0 and 1, GATE is always enabled (HIGH). For Timer 2 and 3, GATE is connected to Bit 0 and

6, respectively, of an Internal Control Port (at address 61H) of the 82380. After a hardware reset, the state of GATE of Timer 2 and 3 is disabled (LOW).

5.3 Modes of Operation

Each timer can be independently programmed to operate in one of six different modes. Timers are programmed by writing a Control Word into the control Word Register followed by an Initial Count (see Programming).

The following are defined for use in describing the different modes of operation.

CLK Pulse—A rising edge, then a falling edge, in that order of CLKIN.

Trigger—A rising edge of a timer's GATE input.

Timer/Counter Loading—The transfer of a count from Count Register (CR) to Count Element (CE).

Note that figures 5-3 through 5-8 show the logical outputs of the timer units, OUT_x . This signal polarity does not reflect that of the $TOUT_x$ signals. See the first paragraph of Section 5.2.2.

5.3.1 MODE 0—INTERRUPT ON TERMINAL COUNT

Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially LOW, and will remain LOW until the counter reaches zero. OUT then goes HIGH and remains HIGH until a new count or a new Mode 0 Control Word is written into the counter.

In this mode, GATE = HIGH enables counting; GATE = LOW disables counting. However, GATE has no effect on OUT.

After the Control Word and initial count are written to a timer, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the

count, so for an initial count of N, OUT does not go HIGH until $N + 1$ CLK pulses after the initial count is written.

If a new count is written to the timer, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1. Writing the first byte disables counting, OUT is set LOW immediately (i.e., no CLK pulse required).
2. Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the counting sequence to be synchronized by software. Again, OUT does not go HIGH until $N + 1$ CLK pulses after the new count of N is written.

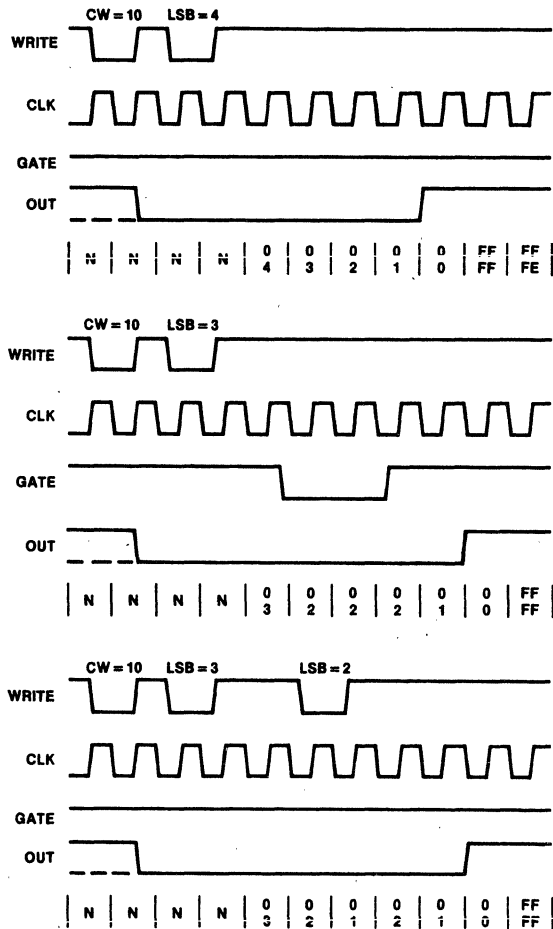
If an initial count is written while GATE is LOW, the counter will be loaded on the next CLK pulse. When GATE goes HIGH, OUT will go HIGH N CLK pulses later; no CLK pulse is needed to load the counter as this has already been done.

5.3.2 MODE 1—GATE RETRIGGERABLE ONE-SHOT

In this mode, OUT will be initially HIGH. OUT will go LOW on the CLK pulse following a trigger to start the one-shot operation. The OUT signal will then remain LOW until the timer reaches zero. At this point, OUT will stay HIGH until the next trigger comes in. Since the state of GATE signals of Timer 0 and 1 are internally set to HIGH.

After writing the Control Word and initial count, the timer is considered 'armed'. A trigger results in loading the timer and setting OUT LOW on the next CLK pulse. Therefore, an initial count of N will result in a one-shot pulse width of N CLK cycles. Note that this one-shot operation is retriggerable; i.e., OUT will remain LOW for N CLK pulses after every trigger. The one-shot operation can be repeated without rewriting the same count into the timer.

If a new count is written to the timer during a one-shot operation, the current one-shot pulse width will not be affected until the timer is retriggered. This is because loading of the new count to CE will occur only when the one-shot is triggered.



290128-68

NOTES:

The following conventions apply to all mode timing diagrams.

1. Counters are programmed for binary (not BCD) counting and for reading/writing least significant byte (LSB) only.
2. The counter is always selected (\overline{CS} always low).
3. CW stands for "Control Word"; CW = 10 means a control word of 10, Hex is written to the counter.
4. LSB stands for "least significant byte" of count.
5. Numbers below diagrams are count values.

The lower number is the least significant byte.

The upper number is the most significant byte. Since the counter is programmed to read/write LSB only, the most significant byte cannot be read.

N stands for an undefined count.

Vertical lines show transitions between count values.

Figure 5-3. Mode 0

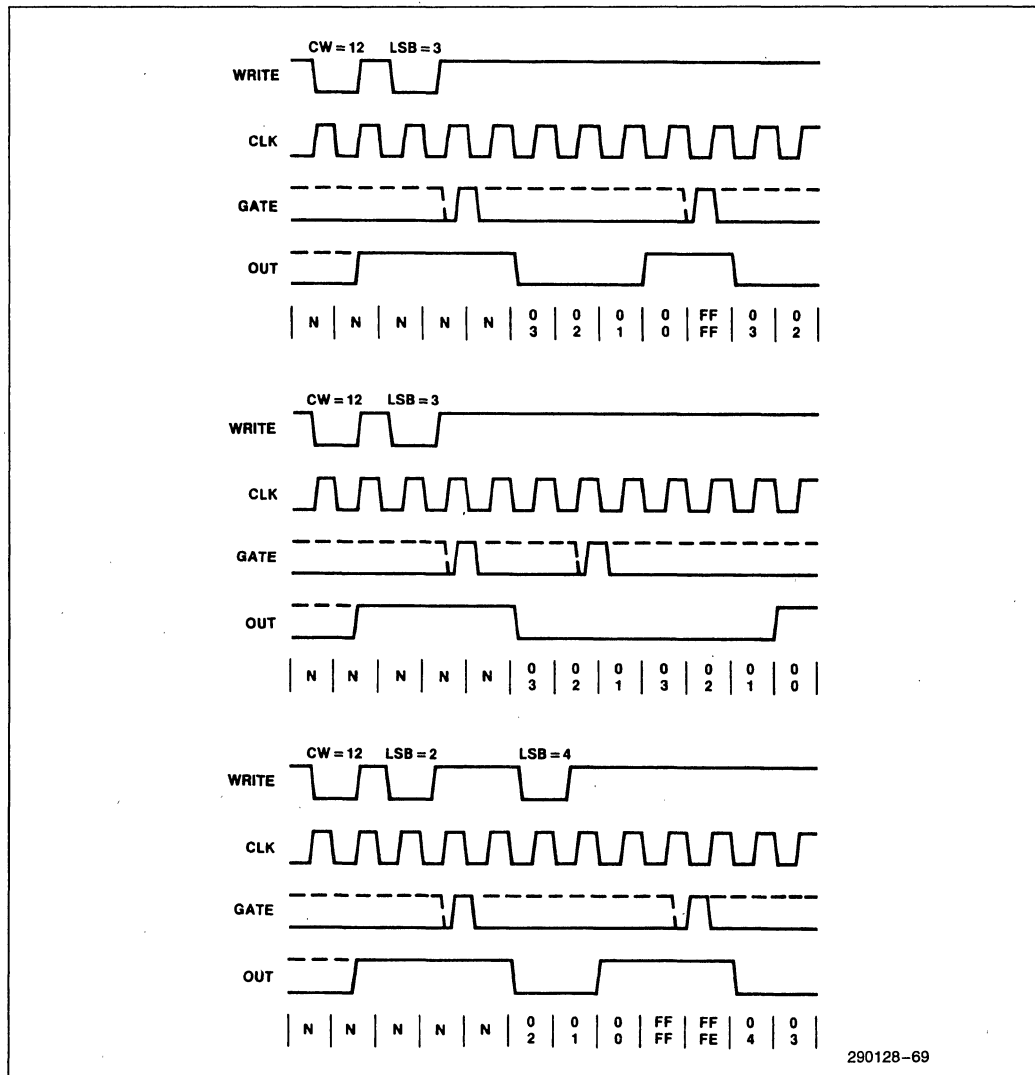


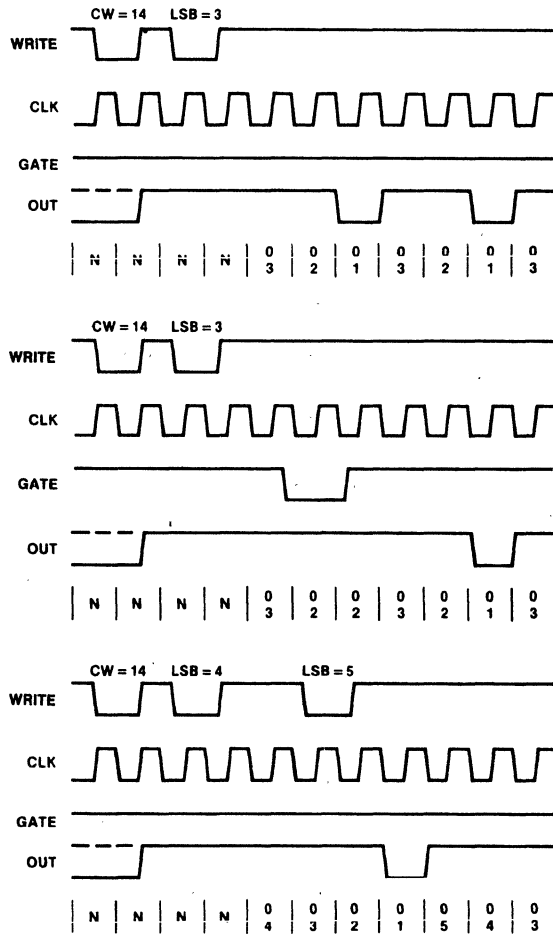
Figure 5-4. Mode 1

5.3.3 MODE 2—RATE GENERATOR

This mode is a divide-by-N counter. It is typically used to generate a Real Time Clock interrupt. OUT will initially be HIGH. When the initial count has decremented to 1, OUT goes LOW for one CLK pulse, then OUT goes HIGH again. Then the timer reloads the initial count and the process is repeated. In other words, this mode is periodic since the same sequence is repeated itself indefinitely. For an initial

count of N, the sequence repeats every N CLK cycles.

Similar to Mode 0, GATE = HIGH enables counting, where GATE = LOW disables counting. If GATE goes LOW during an output pulse (LOW), OUT is set HIGH immediately. A trigger (rising edge on GATE) will reload the timer with the initial count on the next CLK pulse. Then, OUT will go LOW (for one CLK pulse) N CLK pulses after the new trigger. Thus, GATE can be used to synchronize the timer.



290128-70

NOTE:
A GATE transition should not occur one clock prior to terminal count.

Figure 5-5. Mode 2

After writing a Control Word and initial count, the timer will be loaded on the next CLK pulse. OUT goes LOW (for the CLK pulse) N CLK pulses after the initial count is written. This is another way the timer may be synchronized by software.

the timer will be loaded with the new count on the next CLK pulse after the trigger, and counting will continue with the new count.

Writing a new count while counting does not affect the current counting sequence because the new count will not be loaded until the end of the current counting cycle. If a trigger is received after writing a new count but before the end of the current period,

5.3.4 MODE 3—SQUARE WAVE GENERATOR

Mode 3 is typically used for Baud Rate generation. Functionally, this mode is similar to Mode 2 except for the duty cycle of OUT. In this mode, OUT will be initially HIGH. When half of the initial count has expired, OUT goes low for the remainder of the count.

The counting sequence will be repeated, thus this mode is also periodic. Note that an initial count of N results in a square wave with a period of N CLK pulses.

The GATE input can be used to synchronize the timer. GATE = HIGH enables counting; GATE = LOW disables counting. If GATE goes LOW while OUT is LOW, OUT is set HIGH immediately (i.e., no CLK pulse is required). A trigger reloads the timer with the initial count on the next CLK pulse.

After writing a Control Word and initial count, the timer will be loaded on the next CLK pulse. This allows the timer to be synchronized by software.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the timer will be loaded with the new count on the next CLK

pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

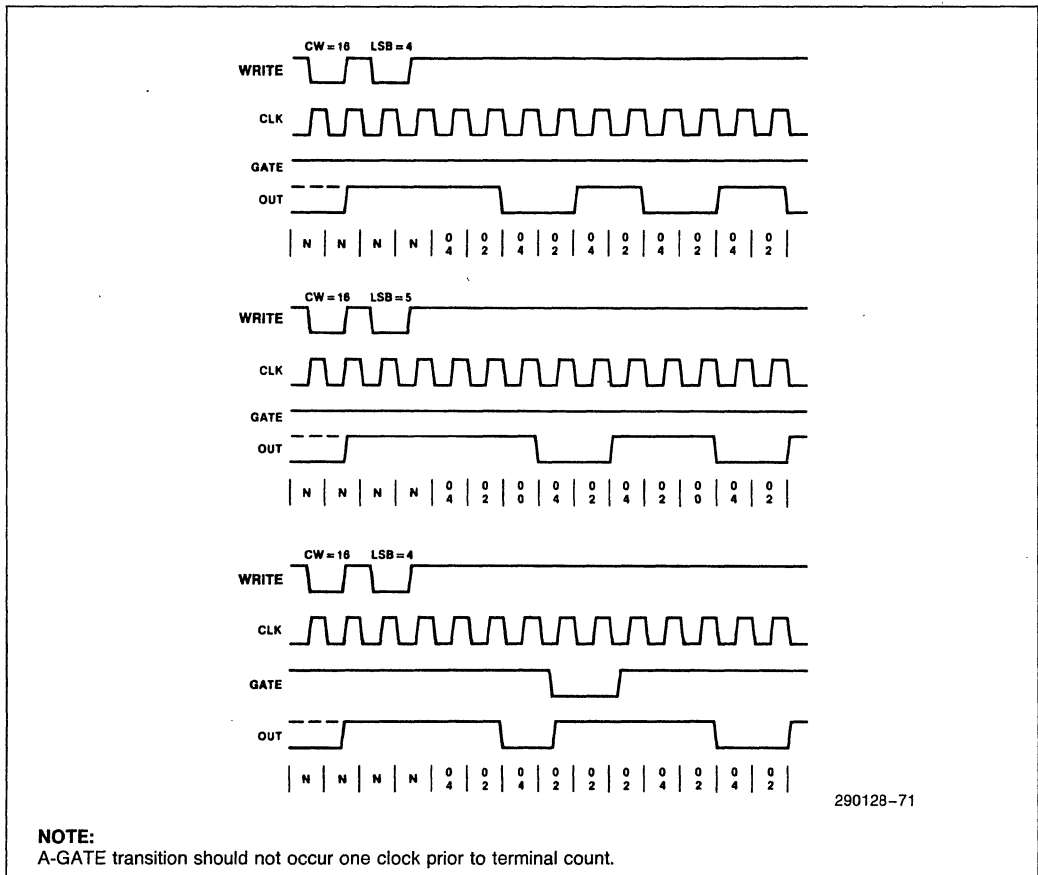
There is a slight difference in operation depending on whether the initial count is EVEN or ODD. The following description is to show exactly how this mode is implemented.

EVEN COUNTS:

OUT is initially HIGH. The initial count is loaded on one CLK pulse and is decremented by two on succeeding CLK pulses. When the count expires (decremented to 2), OUT changes to LOW and the timer is reloaded with the initial count. The above process is repeated indefinitely.

ODD COUNTS:

OUT is initially HIGH. The initial count minus one (which is an even number) is loaded on one CLK



NOTE:
A-GATE transition should not occur one clock prior to terminal count.

290128-71

Figure 5-6. Mode 3

pulse and is decremented by two on succeeding CLK pulses. One CLK pulse after the count expires (decremented to 2), OUT goes LOW and the timer is loaded with the initial count minus one again. Succeeding CLK pulses decrement the count by two. When the count expires, OUT goes HIGH immediately and the timer is reloaded with the initial count minus one. The above process is repeated indefinitely. So for ODD counts, OUT will be HIGH for $(N + 1)/2$ counts and LOW for $(N - 1)/2$ counts.

5.3.5 MODE 4—INITIAL COUNT TRIGGERED STROBE

This mode allows a strobe pulse to be generated by writing an initial count to the timer. Initially, OUT will

be HIGH. When a new initial count is written into the timer, the counting sequence will begin. When the initial count expires (decremented to 1), OUT will go LOW for one CLK pulse and then go HIGH again.

Again, GATE = HIGH enables counting while GATE = LOW disables counting. GATE has no effect on OUT.

After writing the Control Word and initial count, the timer will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe LOW until N + 1 CLK pulses after initial count is written.

If a new count is written during counting, it will be loaded in the next CLK pulse and counting will continue from the new count.

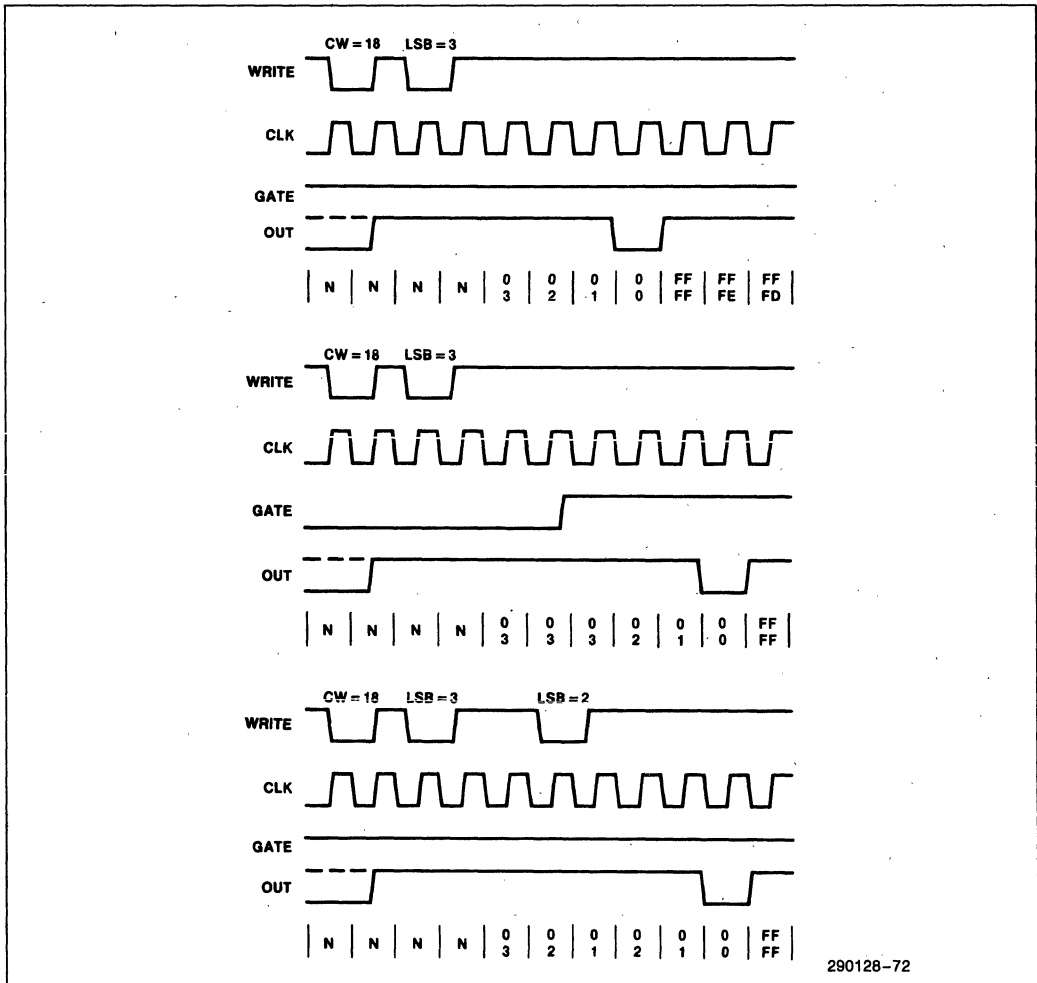


Figure 5-7. Mode 4

If a two-byte count is written, the following will occur:

1. Writing the first byte has no effect on counting.
2. Writing the second byte allows the new count to be loaded on the next CLK pulse.

OUT will strobe LOW $N + 1$ CLK pulses after the new count of N is written. Therefore, when the strobe pulse will occur after a trigger depends on the value of the initial count loaded.

5.3.6 MODE 5—GATE RETRIGGERABLE STROBE

Mode 5 is very similar to Mode 4 except the count sequence is triggered by the GATE signal instead of

by writing an initial count. Initially, OUT will be HIGH. Counting is triggered by a rising edge of GATE. When the initial count has expired (decremented to 1), OUT will go LOW for one CLK pulse and then go HIGH again.

After loading the Control Word and initial count, the Count Element will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count. Therefore, for an initial count of N , OUT does not strobe LOW until $N + 1$ CLK pulses after a trigger.

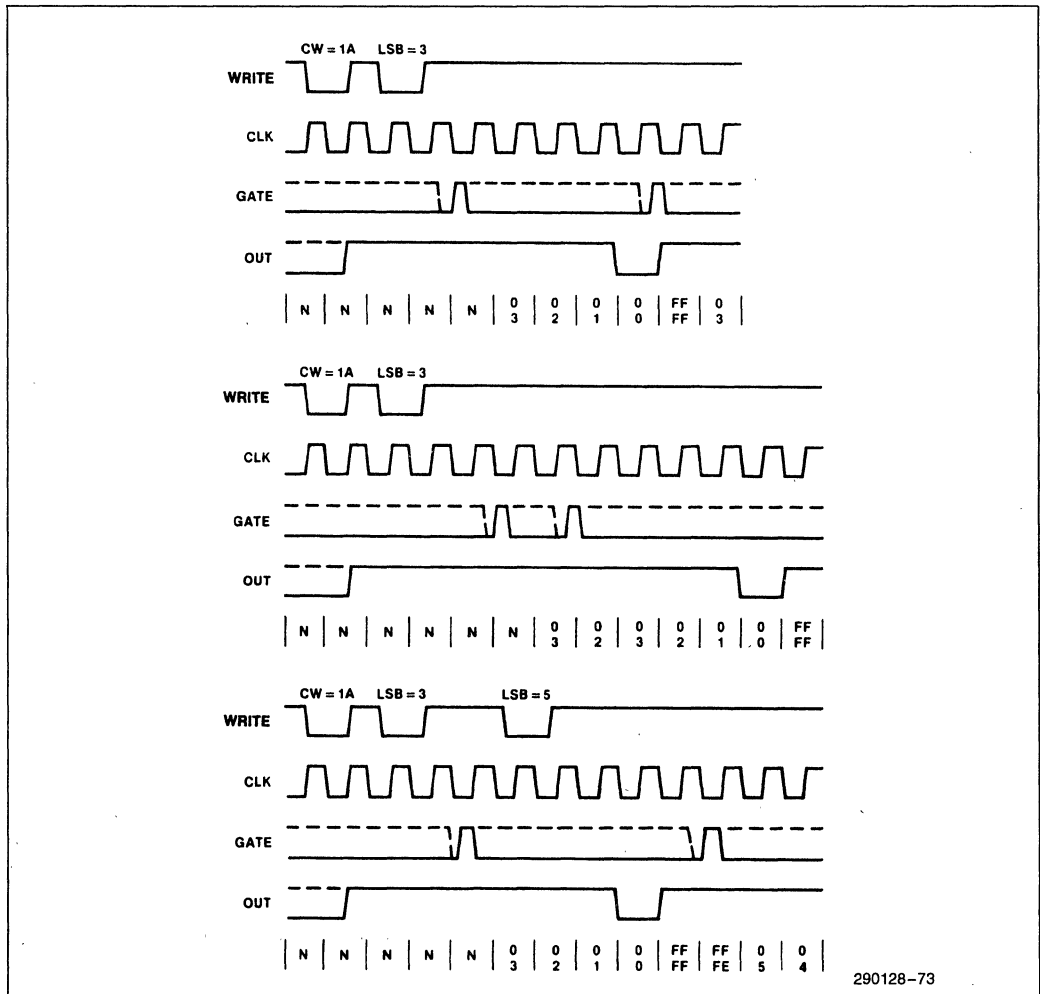


Figure 5-8. Mode 5

SUMMARY OF GATE OPERATIONS

Mode	GATE LOW or Going LOW	GATE Rising	GATE HIGH
0	Disable Count	No Effect	Enable Count
1	No Effect	1. Initiate Count 2. Reset Output After Next Clock	No Effect
2	1. Disable Count 2. Sets Output HIGH Immediately	Initiate Count	Enable Count
3	1. Disable Count 2. Sets Output HIGH Immediately	Initiate Count	Enable Count
4	Disable Count	No Effect	Enable Count
5	No Effect	Initiate Count	No Effect

The counting sequence is retriggerable. Every trigger will result in the timer being loaded with the initial count on the next CLK pulse.

If the new count is written during counting, the current counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the timer will be loaded with the new count on the next CLK pulse and a new count sequence will start from there.

5.3.7 OPERATION COMMON TO ALL MODES

5.3.7.1 GATE

The GATE input is always sampled on the rising edge of CLKIN. In Modes 0, 2, 3 and 4, the GATE input is level sensitive. The logic level is sampled on the rising edge of CLKIN. In Modes 1, 2, 3 and 5, the GATE input is rising edge sensitive. In these modes, a rising edge of GATE (trigger) sets an edge sensitive flip-flop in the timer. The flip-flop is reset immediately after it is sampled. This way, a trigger will be detected no matter when it occurs; i.e., a HIGH logic level does not have to be maintained until the next rising edge of CLKIN. Note that in Modes 2 and 3, the GATE input is both edge and level sensitive.

5.3.7.2 Counter

New counts are loaded and counters are decremented on the falling edge of CLKIN. The largest possible initial count is 0. This is equivalent to 2**16 for binary counting and 10**4 for BCD counting.

Note that the counter does not stop when it reaches zero. In Modes 0, 1, 4, and 5, the counter 'wraps

around' to the highest count: either FFFF Hex for binary counting or 9999 for BCD counting, and continues counting. Modes 2 and 3 are periodic. The counter reloads itself with the initial count and continues counting from there.

The minimum and maximum initial count in each counter depends on the mode of operation. They are summarized below.

Mode	Min	Max
0	1	0
1	1	0
2	2	0
3	2	0
4	1	0
5	1	0

5.4 Register Set Overview

The Programmable Interval Timer module of the 82380 contains a set of six registers. The port address map of these registers is shown in Table 5-2.

Table 5-2. Timer Register Port Address Map

Port Address	Description
40H	Counter 0 Register (read/write)
41H	Counter 1 Register (read/write)
42H	Counter 2 Register (read/write)
43H	Control Word Register I (Counter 0, 1 & 2) (write-only)
44H	Counter 3 Register (read/write)
45H	Reserved
46H	Reserved
47H	Control Word Register II (Counter 3) (write-only)

5.4.1 COUNTER 0, 1, 2, 3 REGISTERS

These four 8-bit registers are functionally identical. They are used to write the initial count value into the respective timer. Also, they can be used to read the latched count value of a timer. Since they are 8-bit registers, reading and writing of the 16-bit initial count must follow the count format specified in the Control Word Registers; i.e., least significant byte only, most significant byte only, or least significant byte then most significant byte (see Programming).

5.4.2 CONTROL WORD REGISTER I & II

There are two Control Word Registers associated with the Timer section. One of the two registers (Control Word Register I) is used to control the operations of Counters 0, 1, and 2 and the other (Control Word Register II) is for Counter 3. The major functions of both Control Word Registers are listed below:

- Select the timer to be programmed.
- Define which mode the selected timer is to operate in.
- Define the count sequence; i.e., if the selected timer is to count as a Binary Counter or a Binary Coded Decimal (BCD) Counter.
- Select the byte access sequence during timer read/write operations; i.e., least significant byte only, most significant byte only, or least significant byte first, then most significant byte.

Also, the Control Word Registers can be programmed to perform a Counter Latch Command or a Read Back Command which will be described later.

5.5 Programming

5.5.1 INITIALIZATION

Upon power-up or reset, the state of all timers is undefined. The mode, count value, and output of all timers are random. From this point on, how each timer operates is determined solely by how it is programmed. Each timer must be programmed before it can be used. Since the outputs of some timers can generate interrupt signals to the 82380, all timers should be initialized to a known state.

Timers are programmed by writing a Control Word into their respective Control Word Registers. Then, an Initial Count can be written into the correspond-

ing Count Register. In general, the programming procedure is very flexible. Only two conventions need to be remembered:

1. For each timer, the Control Word must be written before the initial count is written.
2. The 16-bit initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte first, followed by most significant byte).

Since the two Control Word Registers and the four Counter Registers have separate addresses, and each timer can be individually selected by the appropriate Control Word Register, no special instruction sequence is required. Any programming sequence that follows the conventions above is acceptable.

A new initial count may be written to a timer at any time without affecting the timer's programmed mode in any way. Count sequence will be affected as described in the Modes of Operation section. Note that the new count must follow the programmed count format.

If a timer is previously programmed to read/write two-byte counts, the following precaution applies. A program must not transfer control between writing the first and second byte to another routine which also writes into the same timer. Otherwise, the read/write will result in incorrect count.

Whenever a Control Word is written to a timer, all control logic for that timer(s) is immediately reset (i.e., no CLK pulse is required). Also, the corresponding output pin, TOUT(#), goes to a known initial state.

5.5.2 READ OPERATION

Three methods are available to read the current count as well as the status of each timer. They are: Read Counter Registers, Counter Latch Command and Read Back Command. Following is a description of these methods.

READ COUNTER REGISTERS

The current count of a timer can be read by performing a read operation on the corresponding Counter Register. The only restriction of this read operation is that the CLKIN of the timers must be inhibited by

using external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result. Note that since all four timers are sharing the same CLKIN signal, inhibiting CLKIN to read a timer will unavoidably disable the other timers also. This may prove to be impractical. Therefore, it is suggested that either the Counter Latch Command or the Read Back Command be used to read the current count of a timer.

Another alternative is to temporarily disable a timer before reading its Counter Register by using the GATE input. Depending on the mode of operation, GATE = LOW will disable the counting operation. However, this option is available on Timer 2 and 3 only, since the GATE signals of the other two timers are internally enabled all the time.

COUNTER LATCH COMMAND

A Counter Latch Command will be executed whenever a special Control Word is written into a Control Word Register. Two bits written into the Control Word Register distinguish this command from a 'regular' Control Word (see Register Bit Definition). Also, two other bits in the Control Word will select which counter is to be latched.

Upon execution of this command, the selected counter's Output Latch (OL) latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the 80386, or until the timer is reprogrammed. The count is then unlatched automatically and the OL returns to 'following' the Counting Element (CE). This allows reading the contents of the counters 'on the fly' without affecting counting in progress. Multiple Counter Latch Commands may be used to latch more than one counter. Each latched count is held until it is read. Counter Latch Commands do not affect the programmed mode of the timer in any way.

If a counter is latched, and at some time later, it is latched again before the prior latched count is read, the second Counter Latch Command is ignored. The count read will then be the count at the time the first command was issued.

In any event, the latched count must be read according to the programmed format. Specifically, if the timer is programmed for two-byte counts, two bytes must be read. However, the two bytes do not have to be read right after the other. Read/write or programming operations of other timers may be performed between them.

Another feature of this Counter Latch Command is that read and write operations of the same timer may be interleaved. For example, if the timer is programmed for two-byte counts, the following sequence is valid.

1. Read least significant byte.
2. Write new least significant byte.
3. Read most significant byte.
4. Write new most significant byte.

If a timer is programmed to read/write two-byte counts, the following precaution applies. A program must not transfer control between reading the first and second byte to another routine which also reads from that same timer. Otherwise, an incorrect count will be read.

READ BACK COMMAND

The Read Back Command is another special Command Word operation which allows the user to read the current count value and/or the status of the selected timer(s). Like the Counter Latch Command, two bits in the Command Word identify this as a Read Back Command (see Register Bit Definition).

The Read Back Command may be used to latch multiple counter Output Latches (OL's) by selecting more than one timer within a Command Word. This single command is functionally equivalent to several Counter Latch Commands, one for each counter to be latched. Each counter's latched count will be held until it is read by the 80386 or until the timer is reprogrammed. The counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple Read Back commands are issued to the same timer without reading the count, all but the first are ignored; i.e., the count read will correspond to the very first Read Back Command issued.

As mentioned previously, the Read Back Command may also be used to latch status information of the selected timer(s). When this function is enabled, the status of a timer can be read from the Counter Register after the Read Back Command is issued. The status information of a timer includes the following:

1. Mode of timer:
 - This allows the user to check the mode of operation of the timer last programmed.
2. State of TOUT pin of the timer:
 - This allows the user to monitor the counter's output pin via software, possibly eliminating some hardware from a system.

3. Null Count/Count available:

The Null Count Bit in the status byte indicates if the last count written to the Count Register (CR) has been loaded into the Counting Element (CE). The exact time this happens depends on the mode of the timer and is described in the Programming section. Until the count is loaded into the Counting Element (CE), it cannot be read from the timer. If the count is latched or read before this occurs, the count value will not reflect the new count just written.

If multiple status latch operations of the timer(s) are performed without reading the status, all but the first command are ignored; i.e., the status read in will correspond to the first Read Back Command issued.

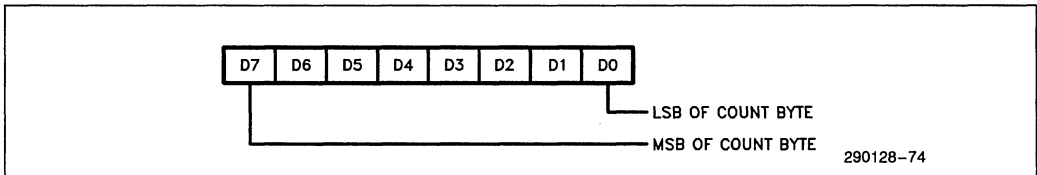
Both the current count and status of the selected timer(s) may be latched simultaneously by enabling both functions in a single Read Back Command. This is functionally the same as issuing two separate Read Back Commands at once. Once again, if multiple read commands are issued to latch both the count and status of a timer, all but the first command will be ignored.

If both count and status of a timer are latched, the first read operation of that timer will return the latched status, regardless of which was latched first. The next one or two (if two count bytes are to be read) read operations return the latched count. Note that subsequent read operations on the Counter Register will return the unlatched count (like the first read method discussed).

5.6 Register Bit Definitions

COUNTER 0, 1, 2, 3 REGISTER (READ/WRITE)

Port Address	Description
40H	Counter 0 Register (read/write)
41H	Counter 1 Register (read/write)
42H	Counter 2 Register (read/write)
44H	Counter 3 Register (read/write)
45H	Reserved
46H	Reserved

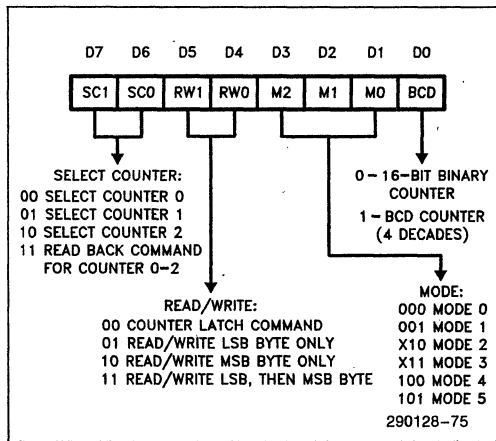


Note that these 8-bit registers are for writing and reading of one byte of the 16-bit count value, either the most significant or the least significant byte.

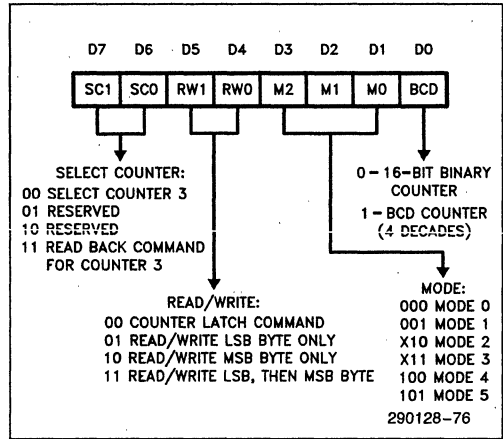
CONTROL WORD REGISTER I & II (WRITE-ONLY)

Port Address	Description
43H	Control Word Register I (Counter 0, 1, 2) (write-only)
47H	Control Word Register II (Counter 3) (write-only)

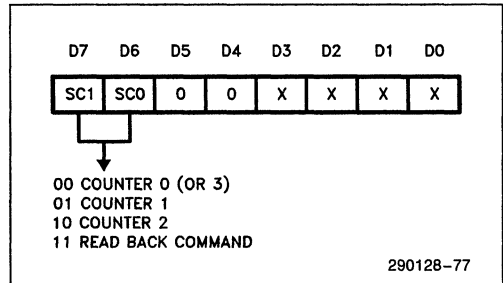
CONTROL WORD REGISTER I



CONTROL WORD REGISTER II



COUNTER LATCH COMMAND FORMAT (Write to Control Word Register)

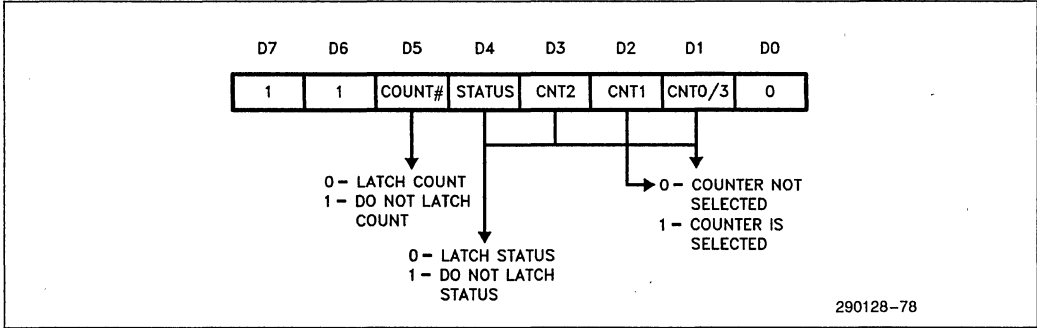


Mode	Timer				Gate Trigger	
	0	1	2	3	Edge	Level
0						X
1	NA	NA	⊙	⊙	X	X
2					X	X
3					X	X
4						X
5	NA	NA	⊙	⊙	X	

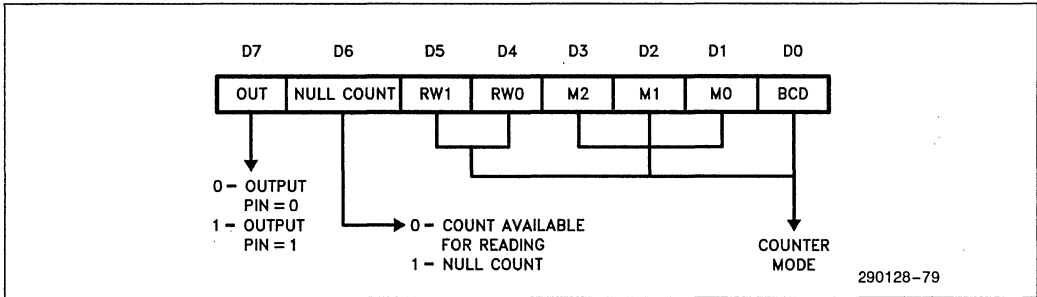
Interrupt on Terminal Count
 Gate Retriggerable One Shot
 Rate Generator
 Square Wave Generator
 Initial Count Triggered Strobe
 Gate Retriggerable Strobe

⊙ = Must use Port 61 to generate edge.
 NA = Not Applicable

READ BACK COMMAND FORMAT
(Write to Control Word Register)



STATUS FORMAT
(Returned from Read Back Command)



6.0 WAIT STATE GENERATOR

6.1 Functional Description

The 82380 contains a programmable Wait State Generator which can generate a pre-programmed number of wait states during both CPU and DMA initiated bus cycles. This Wait State Generator is capable of generating 1 to 16 wait states in non-pipe-

lined mode, and 0 to 15 wait states in pipelined mode. Depending on the bus cycle type and the two Wait State Control inputs (WSC 0-1), a pre-programmed number of wait states in the selected Wait State Register will be generated.

The Wait State Generator can also be disabled to allow the use of devices capable of generating their own READY# signals. Figure 6-1 is a block diagram of the Wait State Generator.

6.2 Interface Signals

The following describes the interface signals which affect the operation of the Wait State Generator. The READY#, WSC0 and WSC1 signals are inputs. READY# is the ready output signal to the host processor.

6.2.1 READY#

READY# is an active LOW input signal which indicates to the 82380 the completion of a bus cycle. In the Master mode (e.g., 82380 initiated DMA transfer), this signal is monitored to determine whether a peripheral or memory needs wait states inserted in the current bus cycle. In the Slave mode, it is used (together with the ADS# signal) to trace CPU bus cycles to determine if the current cycle is pipelined.

6.2.2 READYO#

READYO# (Ready Out#) is an active LOW output signal and is the output of the Wait State Generator. The number of wait states generated depends on the WSC(0-1) inputs. Note that special cases are

handled for access to the 82380 internal registers and for the Refresh cycles. For 82380 internal register access, READYO# will be delayed to take into account the command recovery time of the register. One or more wait states will be generated in a pipelined cycle. During refresh, the number of wait states will be determined by the preprogrammed value in the Refresh Wait State Register.

In the simplest configuration, READYO# can be connected to the READY# input of the 82380 and the 80386 CPU. This is, however, not always the case. If external circuitry is to control the READY# inputs as well, additional logic will be required (see Application Issues).

6.2.3 WSC(0-1)

These two Wait State Control inputs select one of the three pre-programmed 8-bit Wait State Registers which determines the number of wait states to be generated. The most significant half of the three Wait State Registers corresponds to memory accesses, the least significant half to I/O accesses. The combination WSC(0-1) = 11 disables the Wait State Generator.

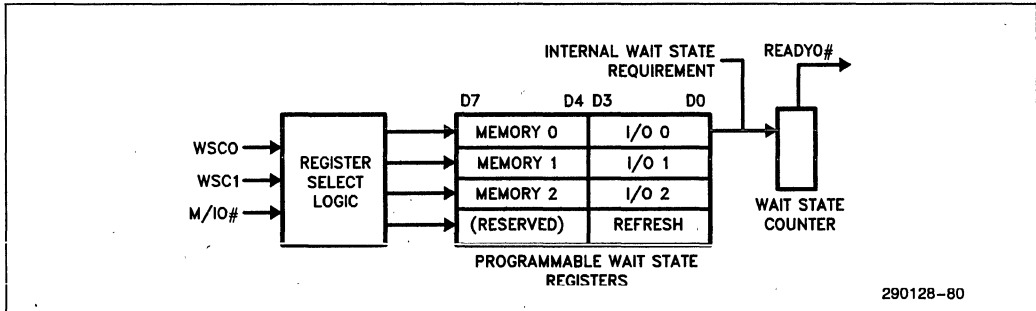


Figure 6-1. Wait State Generator Block Diagram

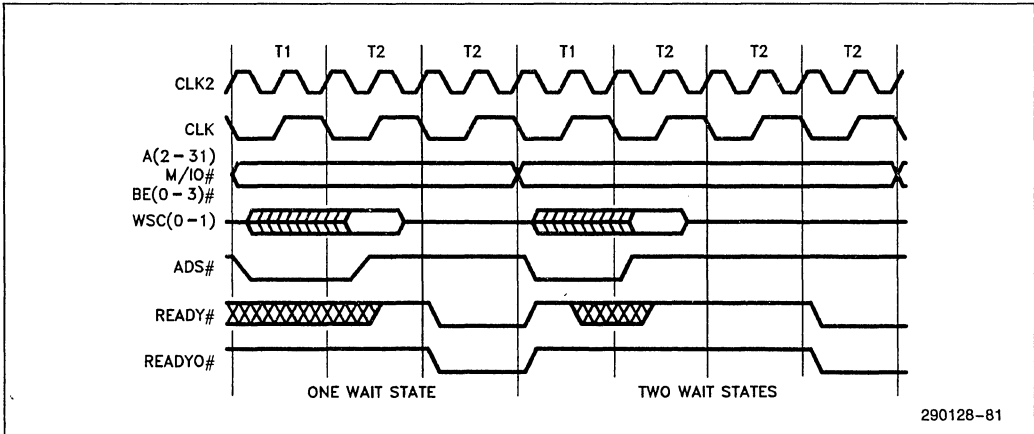


Figure 6-2. Wait States in Non-Pipelined Cycles

290128-81

6.3 Bus Function

6.3.1 WAIT STATES IN NON-PIPELINED CYCLE

The timing diagram of two typical non-pipelined cycles with 82380 generated wait states is shown in Figure 6-2. In this diagram, it is assumed that the internal registers of the 82380 are not addressed. During the first T2 state of each bus cycle, the Wait State Control and the M/IO# inputs are sampled to determine which Wait State Register (if any) is selected. If the WSC inputs are active (i.e., not both are driven HIGH), the pre-programmed number of wait states corresponding to the selected Wait State Register will be requested. This is done by driving the READYO# output HIGH during the end of each T2 state.

The WSC(0-1) inputs need only be valid during the very first T2 state of each non-pipelined cycle. As a general rule, the WSC inputs are sampled on the

rising edge of the next clock (82384 CLK) after the last state when ADS# (Address Status) is asserted.

The number of wait states generated depends on the type of bus cycle, and the number of wait states requested. The various combinations are discussed below.

1. Access the 82380 internal registers: 2 to 5 wait states, depending upon the specific register addressed. Some back-to-back sequences to the Interrupt Controller will require 7 wait states.
2. Interrupt Acknowledge to the 82380: 5 wait states.
3. Refresh: As programmed in the Refresh Wait State Register (see Register Set Overview). Note that if WSC(0-1) = 11, READYO# will stay inactive.
4. Other bus cycles: Depending on WSC(0-1) and M/IO# inputs, these inputs select a Wait State Register in which the number of wait states will be equal to the pre-programmed wait state count in the register plus 1. The Wait State Register selection is defined as follows (Table 6-1).

Table 6-1. Wait State Register Selection

M/IO#	WSC(1-0)	Register Selected
0	00	WAIT REG 0 (I/O half)
0	01	WAIT REG 1 (I/O half)
0	10	WAIT REG 2 (I/O half)
1	00	WAIT REG 0 (MEM half)
1	01	WAIT REG 1 (MEM half)
1	10	WAIT REG 2 (MEM half)
X	11	Wait State Gen. Disabled

The Wait State Control signals, WSC(0-1), can be generated with the address decode and the Read/Write control signals as shown in Figure 6-3.

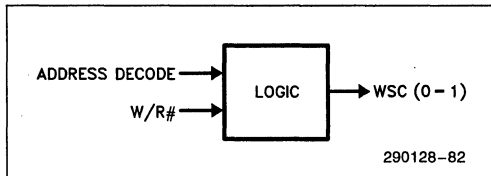


Figure 6-3. WSC(0-1) Generation

Note that during HALT and SHUTDOWN, the number of wait states will depend on the WSC(0-1) inputs, which will select the memory half of one of the Wait State Registers (see CPU Reset and Shutdown Detect).

6.3.2 WAIT STATES IN PIPELINED CYCLE

The timing diagram of two typical pipelined cycles with 82380 generated wait states is shown in Figure 6-4. Again, in this diagram, it is assumed that the 82380 internal registers are not addressed. As defined in the timing of the 80386 processor, the Address (A 2-31), Byte Enable (BE 0-3), and other control signals (M/IO#, ADS#) are asserted one T state earlier than in a non-pipelined cycle; i.e., they are asserted at T2P. Similar to the non-pipelined case, the Wait State Control (WSC) inputs are sampled in the middle of the state after the last state when the ADS# signal is asserted. Therefore, the WSC inputs should be asserted during the T1P state of each pipelined cycle (which is one T state earlier than in the non-pipelined cycle).

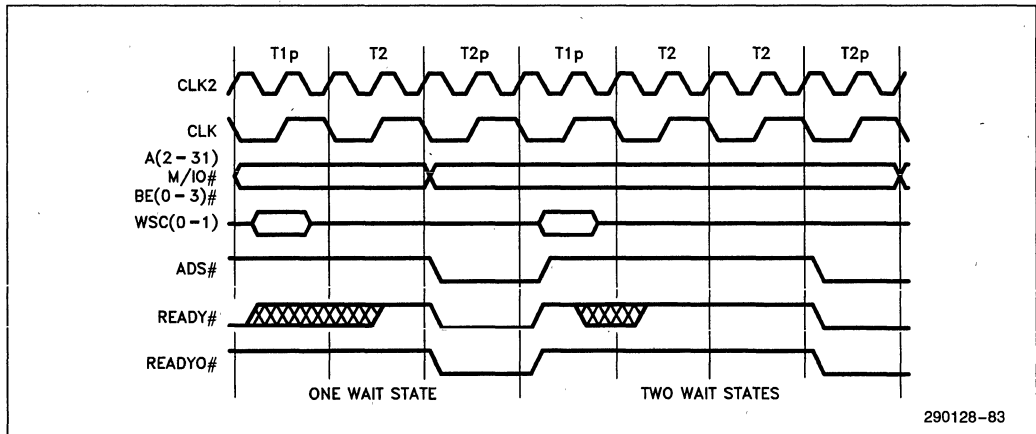


Figure 6-4. Wait State in Pipelined Cycles

The number of wait states generated in a pipelined cycle is selected in a similar manner as in the non-pipelined case discussed in the previous section. The only difference here is that the actual number of wait states generated will be one less than that of the non-pipelined cycle. This is done automatically by the Wait State Generator.

6.3.3 EXTENDING AND EARLY TERMINATING BUS CYCLE

The 82380 allows external logic to either add wait states or cause early termination of a bus cycle by controlling the **READY#** input to the 82380 and the host processor. A possible configuration is shown in Figure 6-5.

The **EXT. RDY#** (External Ready) signal of Figure 6-5 allows external devices to cause early termination of a bus cycle. When this signal is asserted **LOW**, the output of the circuit will also go **LOW** (even though the **READYO#** of the 82380 may still

be **HIGH**). This output is fed to the **READY#** input of the 80386 and the 82380 to indicate the completion of the current bus cycle.

Similarly, the **EXT. NOT READY** (External Not Ready) signal is used to delay the **READY#** input of the processor and the 82380. As long as this signal is driven **HIGH**, the output of the circuit will drive the **READY#** input **HIGH**. This will effectively extend the duration of a bus cycle. However, it is important to note that if the two-level logic is not fast enough to satisfy the **READY#** setup time, the OR gate should be eliminated. Instead, the 82380 Wait State Generator can be disabled by driving both **WSC(0-1)** **HIGH**. In this case, the addressed memory or I/O device should activate the external **READY#** input whenever it is ready to terminate the current bus cycle.

Figure 6-6 and 6-7 show the timing relationships of the ready signals for the early termination and extension of the bus cycles. Section 6.7, Application Issues, contains a detailed timing analysis of the external circuit.

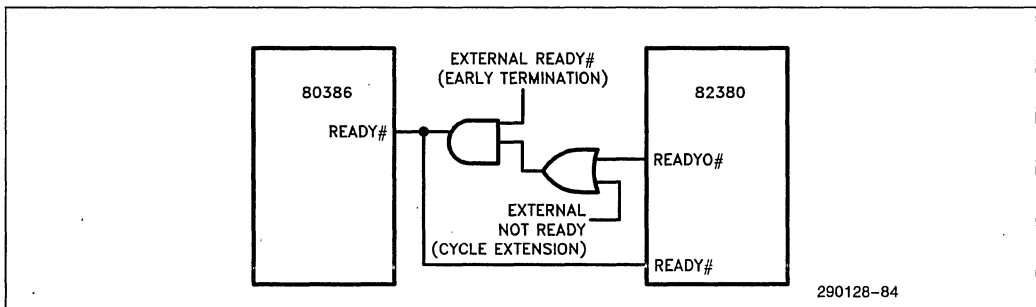


Figure 6-5. External 'READY' Control Logic

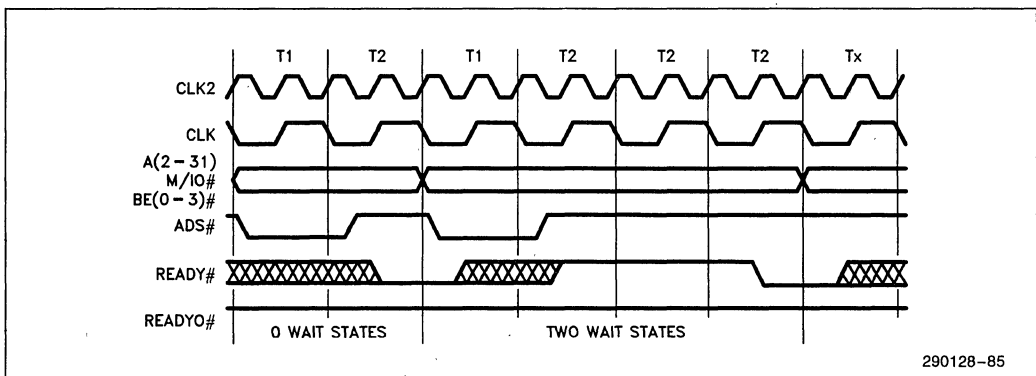


Figure 6-6. Early Termination of Bus Cycle By 'READY#'

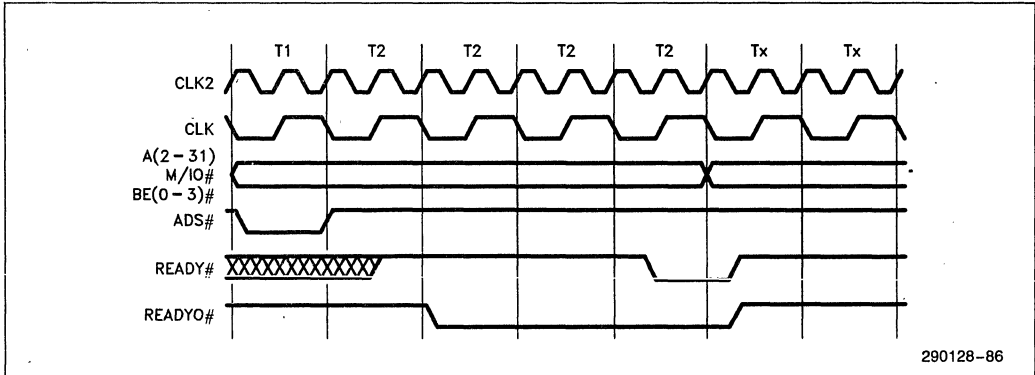


Figure 6-7. Extending Bus Cycle by 'READY#'

290128-86

Due to the following implications, it should be noted that early termination of bus cycles in which 82380 internal registers are accessed is not recommended.

1. Erroneous data may be read from or written into the addressed register.
2. The 82380 must be allowed to recover either before HLDA (Hold Acknowledge) is asserted or before another bus cycle into an 82380 internal register is initiated.

The recovery time, in bus periods, equals the remaining wait states that were avoided plus 4.

6.4 Register Set Overview

Altogether, there are four 8-bit internal registers associated with the Wait State Generator. The port address map of these registers is shown below in Table 6-2. A detailed description of each follows.

Table 6-2. Register Address Map

Port Address	Description
72H	Wait State Reg 0 (read/write)
73H	Wait State Reg 1 (read/write)
74H	Wait State Reg 2 (read/write)
75H	Ref. Wait State Reg (read/write)

WAIT STATE REGISTER 0, 1, 2

These three 8-bit read/write registers are functionally identical. They are used to store the pre-programmed wait state count. One half of each register contains the wait state count for I/O accesses while the other half contains the count for memory accesses. The total number of wait states generated will depend on the type of bus cycle. For a non-pipelined cycle, the actual number of wait states requested is equal to the wait state count plus 1. For a pipelined cycle, the number of wait states will be equal to the wait state count in the selected register. Therefore, the Wait State Generator is capable of generating 1 to 16 wait states in non-pipelined mode, and 0 to 15 wait states in pipelined mode.

Note that the minimum wait state count in each register is 0. This is equivalent to 0 wait states for a pipelined cycle and 1 wait state for a non-pipelined cycle.

REFRESH WAIT STATE REGISTER

Similar to the Wait State Registers discussed above, this 4-bit register is used to store the number of wait states to be generated during the DRAM refresh cycle. Note that the Refresh Wait State Register is not selected by the WSC inputs. It will automatically be

chosen whenever a DRAM refresh cycle occurs. If the Wait State Generator is disabled during the refresh cycle ($WSC(0-1) = 11$), $READY\#$ will stay inactive and the Refresh Wait State Register is ignored.

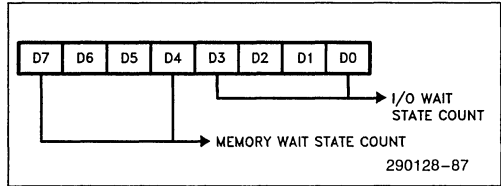
6.5 Programming

Using the Wait State Generator is relatively straightforward. No special programming sequence is required. In order to ensure the expected number of wait states will be generated when a register is selected, the registers to be used must be programmed after power-up by writing the appropriate wait state count into each register. Note that upon hardware reset, all Wait State Registers are initialized with the value FFH, giving the maximum number of wait states possible. Also, each register can be read to check the wait state count previously stored in the register.

6.6 Register Bit Definition

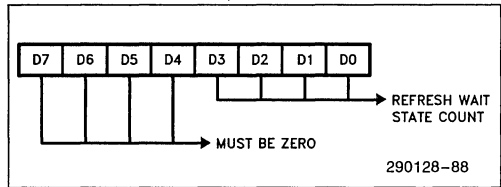
WAIT STATE REGISTER 0, 1, 2

Port Address	Description
72H	Wait State Register 0 (read/write)
73H	Wait State Register 1 (read/write)
74H	Wait State Register 2 (read/write)



REFRESH WAIT STATE REGISTER

Port Address: 75H (Read/Write)



6.7 Application Issues

6.7.1 EXTERNAL 'READY' CONTROL LOGIC

As mentioned in section 6.3.3, wait state cycles generated by the 82380 can be terminated early or extended longer by means of additional external logic (see Figure 6-5). In order to ensure that the $READY\#$ input timing requirement of the 80386 and the 82380 is satisfied, special care must be taken when designing this external control logic. This section addresses the design requirements.

A simplified block diagram of the external logic along with the READY# timing diagram is shown in Figure 6-8. The purpose is to determine the maximum delay time allowed in the external control logic in order to satisfy the READY# setup time.

First, it will be assumed that the 80386 is running at 16 MHz (i.e., CLK2 and 32 MHz). Therefore, one bus state (two CLK2 periods) will be equivalent to 62.5 nsec. According to the AC specifications of the

82380, the maximum delay time for valid READYO# signal is 31 ns after the rising edge of CLK2 in the beginning of T2 (for non-pipelined cycle) or T2P (for pipelined cycle). Also, the minimum READY# setup time of the 80386 and the 82380 should be 20 ns before the rising edge of CLK2 at the beginning of the next bus state. This limits the total delay time for the external READY# control logic to be 11 ns (62.5-31-21) in order to meet the READY# setup timing requirement.

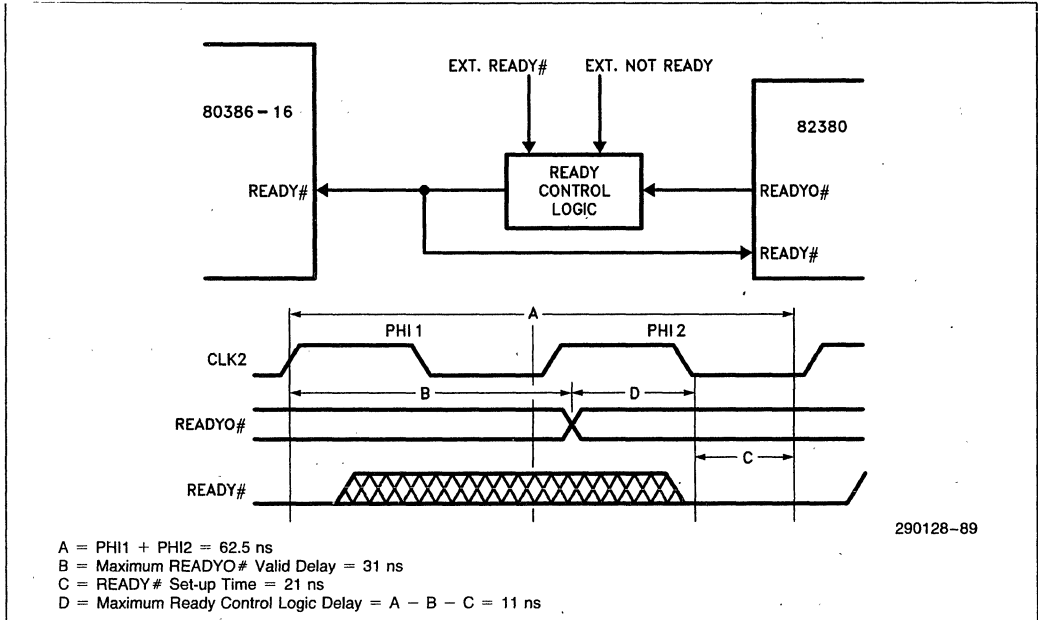


Figure 6-8. 'READY' Timing Consideration

7.0 DRAM REFRESH CONTROLLER

7.1 Functional Description

The 82380 DRAM Refresh Controller consists of a 24-bit Refresh Address Counter and Refresh Request logic for DRAM refresh operations (see Figure 7-1). TIMER 1 can be used as a trigger signal to the DRAM Refresh Request logic. The Refresh Bus Size can be programmed to be 8-, 16-, or 32-bit wide. Depending on the Refresh Bus Size, the Refresh Address Counter will be incremented with the appropriate value after every refresh cycle. The internal logic of the 82380 will give the Refresh operation the highest priority in the bus control arbitration process. Bus control is not released and re-requested if the 82380 is already a bus master.

7.2 Interface Signals

7.2.1 TOUT1/REF

The dual function output pin of TIMER 1 (TOUT1/REF #) can be programmed to generate DRAM Refresh signal. If this feature is enabled, the rising edge of TIMER 1 output (TOUT1) will trigger the DRAM Refresh Request logic. After some delay for gaining access of the bus, the 82380 DRAM Controller will generate a DRAM Refresh signal by driving REF # output LOW. This signal is cleared after the refresh cycle has taken place, or by a hardware reset.

If the DRAM Refresh feature is disabled, the TOUT1/REF # output pin is simply the TIMER 1 output. Detailed information of how TIMER 1 operates is discussed in section 6—Programmable Interval Timer, and will not be repeated here.

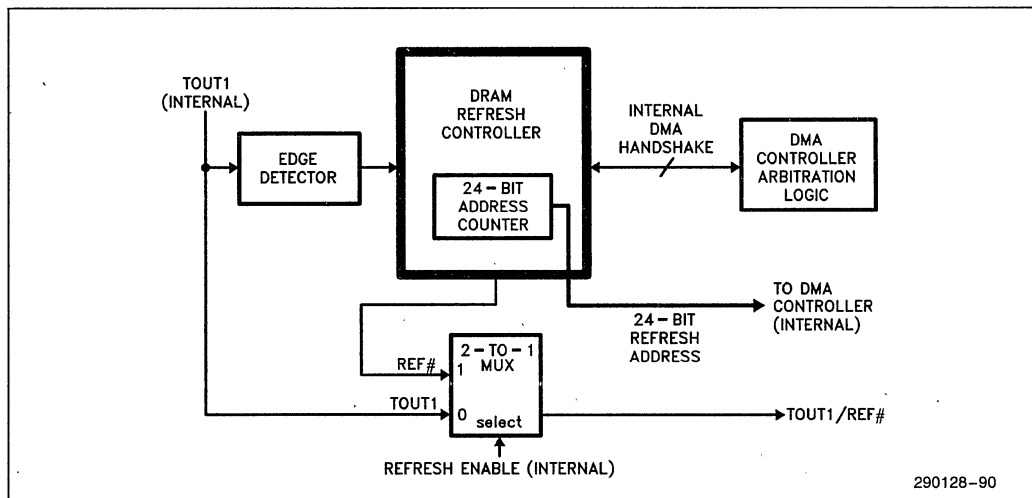


Figure 7-1. DRAM Refresh Controller

7.3 Bus Function

7.3.1 ARBITRATION

In order to ensure data integrity of the DRAMs, the 82380 gives the DRAM Refresh signal the highest priority in the arbitration logic. It allows DRAM Refresh to interrupt a DMA in progress in order to perform the DRAM Refresh cycle. The DMA service will be resumed after the refresh is done.

In case of a DRAM Refresh during a DMA process, the cascaded device will be requested to get off the bus. This is done by deasserting the EDACK signal. Once DREQn goes inactive, the 82380 will perform the refresh operation. Note that the DMA controller does not completely relinquish the system bus during refresh. The Refresh Generator simply 'steals' a bus cycle between DMA accesses.

Figure 7-2 shows the timing diagram of a Refresh Cycle. Upon expiration of TIMER 1, the 82380 will try to take control of the system bus by asserting HOLD. As soon as the 82380 see HLDA go active, the DRAM Refresh Cycle will be carried out by activating the REF# signal as well as the refresh address and control signals on the system bus (Note

that REF# will not be active until two CLK periods after HLDA is asserted). The address bus will contain the 24-bit address currently in the Refresh Address Counter. The control signals are driven the same way as in a Memory Read cycle. This 'read' operation is complete when the READY# signal is driven LOW. Then, the 82380 will relinquish the bus by de-asserting HOLD. Typically, a Refresh Cycle without wait states will take five bus states to execute. If 'n' wait states are added, the Refresh Cycle will last for five plus 'n' bus states.

How often the Refresh Generation will initiate a refresh cycle depends on the frequency of CLKIN as well as TIMER1's programmed mode of operation. For this specific application, TIMER1 should be programmed to operate in Mode 2 or 3 to generate a constant clock rate. See section 6—Programmable Interval Timer for more information on programming the timer. One DRAM Refresh Cycle will be generated each time TIMER 1 expires (when TOUT1 changes to LOW to HIGH).

The Wait State Generator can be used to insert wait states during a refresh cycle. The 82380 will automatically insert the desired number of wait states as programmed in the Refresh Wait State Register (see Wait State Generator).

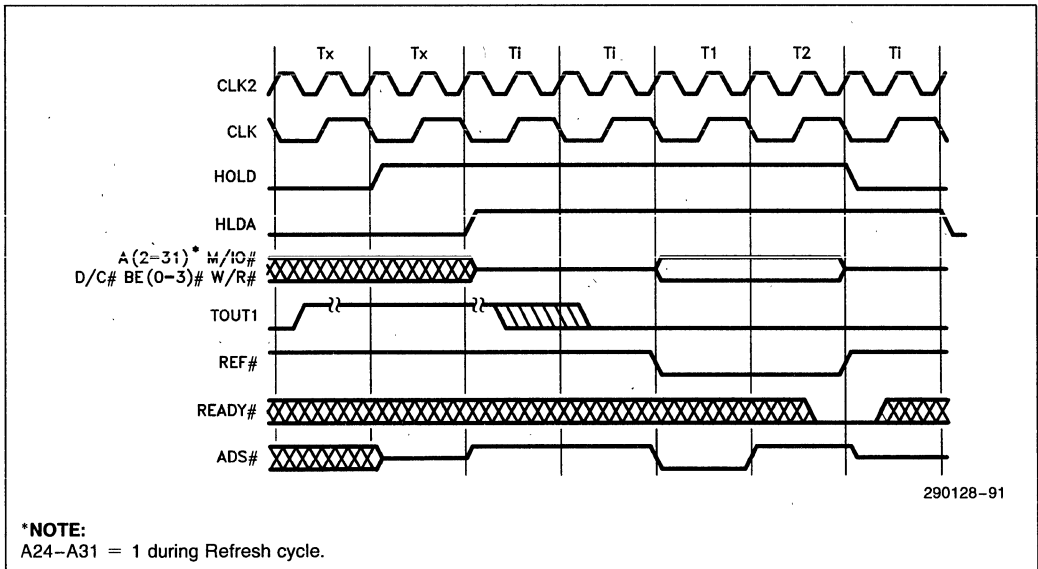


Figure 7-2. 82380 Refresh Cycle

7.4 Modes of Operation

7.4.1 WORD SIZE AND REFRESH ADDRESS COUNTER

The 82380 supports 8-, 16- and 32-bit refresh cycle. The bus width during a refresh cycle is programmable (see Programming). The bus size can be programmed via the Refresh Control Register (see Register Overview). If the DRAM bus size is 8-, 16-, or 32-bits, the Refresh Address Counter will be incremented by 1, 2, or 4, respectively.

The Refresh Address Counter is cleared by a hardware reset.

7.5 Register Set Overview

The Refresh Generator has two internal registers to control its operation. They are the Refresh Control Register and the Refresh Wait State Register. Their port address map is shown in Table 7-1 below.

Port Address	Description
1CH	Refresh Control Reg. (read/write)
75H	Ref. Wait State Reg. (read/write)

Table 7-1. Register Address Map

The Refresh Wait State Register is not part of the Refresh Generator. It is only used to program the number of wait states to be inserted during a refresh cycle. This register is discussed in detail in section 7 (Wait State Generator) and will not be repeated here.

REFRESH CONTROL REGISTER

This 2-bit register serves two functions. First, it is used to enable/disable the DRAM Refresh function output. If disabled, the output of TIMER 1 is simply used as a general purpose timer. The second function of this register is to program the DRAM bus size for the refresh operation. The programmed bus size also determines how the Refresh Address Counter will be incremented after each refresh operation.

7.6 Programming

Upon hardware reset, the DRAM Refresh function is disabled (the Refresh Control Register is cleared). The following programming steps are needed before the Refresh Generator can be used. Since the rate of refresh cycles depends on how TIMER 1 is programmed, this timer must be initialized with the desired mode of operation as well as the correct refresh interval (see Programming Interval Timer).

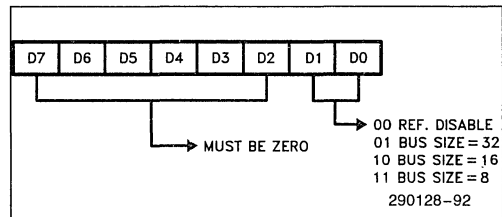
Whether or not wait states are to be generated during a refresh cycle, the Refresh Wait State Register must also be programmed with the appropriate value. Then, the DRAM Refresh feature must be enabled and the DRAM bus width should be defined. These can be done in one step by writing the appropriate control word into the Refresh Control Register (see Register Bit Definition). After these steps are done, the refresh operation will automatically be invoked by the Refresh Generator upon expiration of Timer 1.

In addition to the above programming steps, it should be noted that after reset, although the TOUT1/REF# becomes the Timer 1 output, the state of this pin is undefined. This is because the Timer module has not been initialized yet. Therefore, if this output is used as a DRAM Refresh signal, this pin should be disqualified by external logic until the Refresh function is enabled. One simple solution is to logically AND this output with HLDA, since HLDA should not be active after reset.

7.7 Register Bit Definition

REFRESH CONTROL REGISTER

Port Address: 1CH (Read/Write)



8.0 RELOCATION REGISTER AND ADDRESS DECODE

8.1 Relocation Register

All the integrated peripheral devices in the 82380 are controlled by a set of internal registers. These registers span a total of 256 consecutive address locations (although not all the 256 locations are used). The 82380 provides a Relocation Register which allows the user to map this set of internal registers into either the memory or I/O address space. The function of the Relocation Register is to define the base address of the internal register set of the 82380 as well as if the registers are to be memory- or I/O-mapped. The format of the Relocation Register is depicted in Figure 8-1.

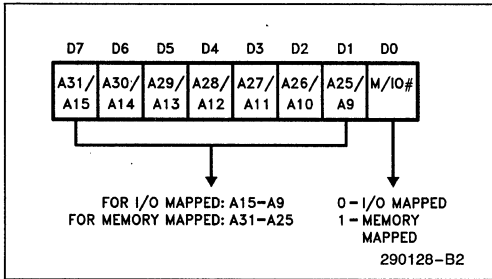


Figure 8-1. Relocation Register

Note that the Relocation Register is part of the internal register set of the 82380. It has a port address of 7FH. Therefore, any time the content of the Relocation Register is changed, the physical location of this register will also be moved. Upon reset of the 82380, the content of the Relocation Register will be cleared. This implies that the 82380 will respond to its I/O addresses in the range of 0000H to 00FFH.

8.1.1 I/O-MAPPED 82380

As shown in the figure, Bit 0 of the Relocation Register determines whether the 82380 registers are to be memory-mapped or I/O-mapped. When Bit 0 is set to '0', the 82380 will respond to I/O Addresses. Address signals BE0#–BE3#, A2–A7 will be used to select one of the internal registers to be accessed. Bit 1 to Bit 7 of the Relocation Register will correspond to A9 to A15 of the Address bus, respectively. Together with A8 implied to be '0', A15 to A8 will be fully decoded by the 82380. The following shows how the 82380 is mapped into the I/O address space.

Example

Relocation Register = 11001110 (0CEH)

82380 will respond to I/O address range from 0CE00H to 0CEFFH.

Therefore, this I/O mapping mechanism allows the 82380 internal registers to be located on any even, contiguous, 256 byte boundary of the system I/O space.

Port Address: 7FH (Read/Write)

8.1.2 MEMORY-MAPPED 82380

When Bit 0 of the Relocation Register is set to '1', the 82380 will respond to memory addresses. Again, Address signals BE0#–BE3#, A2–A7 will be used to select one of the internal registers to be accessed. Bit 1 to Bit 7 of the Relocation Register will correspond to A25–A31, respectively. A24 is assumed to be '0', and A8–A23 are ignored. Consider the following example.

Example

Relocation Register = 10100111 (0A7H)

The 82380 will respond to memory addresses in the range of 0A6XXX00H to 0A6XXXXFFH (where 'X' is don't care).

This scheme implies that the internal register can be located in any even, contiguous, 2**24 byte page of the memory space.

8.2 Address Decoding

As mentioned previously, the 82380 internal registers do not occupy the entire contiguous 256 address locations. Some of the locations are 'unoccupied'. The 82380 always decodes the lower 8 address bits (A0–A7) to determine if any one of its registers is being accessed. If the address does not correspond to any of its registers, the 82380 will not respond. This allows external devices to be located within the 'holes' in the 82380 address space. Note that there are several unused addresses reserved for future Intel peripheral devices.

9.0 CPU RESET AND SHUTDOWN DETECT

The 82380 will activate the CPURST signal to reset the host processor when one of the following conditions occurs:

- 82380 RESET is active;
- 82380 detects a 80386 Shutdown cycle (this feature can be disabled);
- CPURST software command is issued to 80386.

Whenever the CPURST signal is activated, the 82380 will reset its own internal Slave-Bus state machine.

9.1 Hardware Reset

Following a hardware reset, the 82380 will assert its CPURST output to reset the host processor. This output will stay active for as long as the RESET input is active. During a hardware reset, the 82380 internal registers will be initialized as defined in the corresponding functional descriptions.

9.2 Software Reset

CPURST can be generated by writing the following bit pattern into 82380 register location 64H.

D7							D0
1	1	1	1	X	X	X	0

X = Don't Care

The Write operation into this port is considered as an 82380 access and the internal Wait State Generator will automatically determine the required number of wait states. The CPURST will be active following the completion of the Write cycle to this port. This signal will last for 62 CLK2 periods. The 82380 should not be accessed until the CPURST is deactivated.

This internal port is Write-Only and the 82380 will not respond to a Read operation to this location. Also, during a CPU software reset command, the 82380 will reset its Slave-Bus state machine. However, its internal registers remain unchanged. This allows the operating system to distinguish a 'warm' reset by reading any 82380 internal register previously programmed for a non-default value. The Diagnostic registers can be used for this purpose (see Internal Control and Diagnostic Ports).

9.3 Shutdown Detect

The 82380 is constantly monitoring the Bus Cycle Definition signals (M/I0#, D/C#, R/W#) and is able to detect when the 80386 executes a Shutdown bus cycle. Upon detection of a processor shutdown, the 82380 will activate the CPURST output for 62 CLK2 periods to reset the host processor. This signal is generated after the Shutdown cycle is terminated by the READY# signal.

Although the 82380 Wait State Generator will not automatically respond to a Shutdown (or Halt) cycle, the Wait State Control inputs (WSC0, WSC1) can be used to determine the number of wait states in the same manner as other non-82380 bus cycle.

This Shutdown Detect feature can be enabled or disabled by writing a control bit in the Internal Control Port at address 61H (see Internal Control and Diag-

nostic Ports). This feature is disabled upon a hardware reset of the 82380. As in the case of Software Reset, the 82380 will reset its Slave-Bus state machine but will not change any of its internal register contents.

10.0 INTERNAL CONTROL AND DIAGNOSTIC PORTS

10.1 Internal Control Port

The format of the Internal Control Port of the 82380 is shown in Figure 10.1. This Control Port is used to enable/disable the Processor Shutdown Detect mechanism as well as controlling the Gate inputs of the Timer 2 and 3. Note that this is a Write-Only port. Therefore, the 82380 will not respond to a read operation to this port. Upon hardware reset, this port will be cleared; i.e., the Shutdown Detect feature and the Gate inputs of Timer 2 and 3 are disabled.

10.2 Diagnostic Ports

Two 8-bit read/write Diagnostic Ports are provided in the 82380. These are two storage registers and have no effect on the operation of the 82380. They can be used to store checkpoint data or error codes in the power-on sequence and in the diagnostic service routines. As mentioned in CPU RESET AND SHUTDOWN DETECT section, these Diagnostic Ports can be used to distinguish between 'cold' and 'warm' reset. Upon hardware reset, both Diagnostic Ports are cleared. The address map of these Diagnostic Ports is shown in Figure 10-2.

Port	Address
Diagnostic Port 1 (Read/Write)	80H
Diagnostic Port 2 (Read/Write)	88H

Figure 10-2. Address Map of Diagnostic Ports

Port Address: 61H (Write Only)

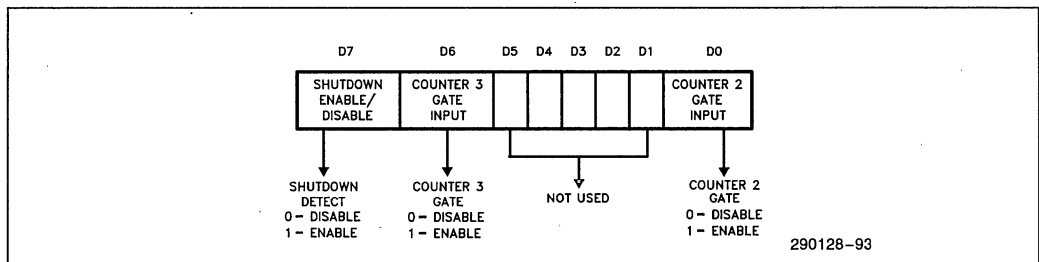


Figure 10-1. Internal Control Port

11.0 INTEL RESERVED I/O PORTS

There are eleven I/O ports in the 82380 address space which are reserved for Intel future peripheral device use only. Their address locations are: 2AH, 3DH, 3EH, 45H, 46H, 76H, 77H, 7DH, 7EH, CCH and CDH. These addresses should not be used in the system since the 82380 may respond to read/write operations to these locations and bus conten-

tion may occur if any peripheral is assigned to the same address location.

12.0 MECHANICAL DATA

12.1 Introduction

In this section, the physical package and its connections are described in detail.

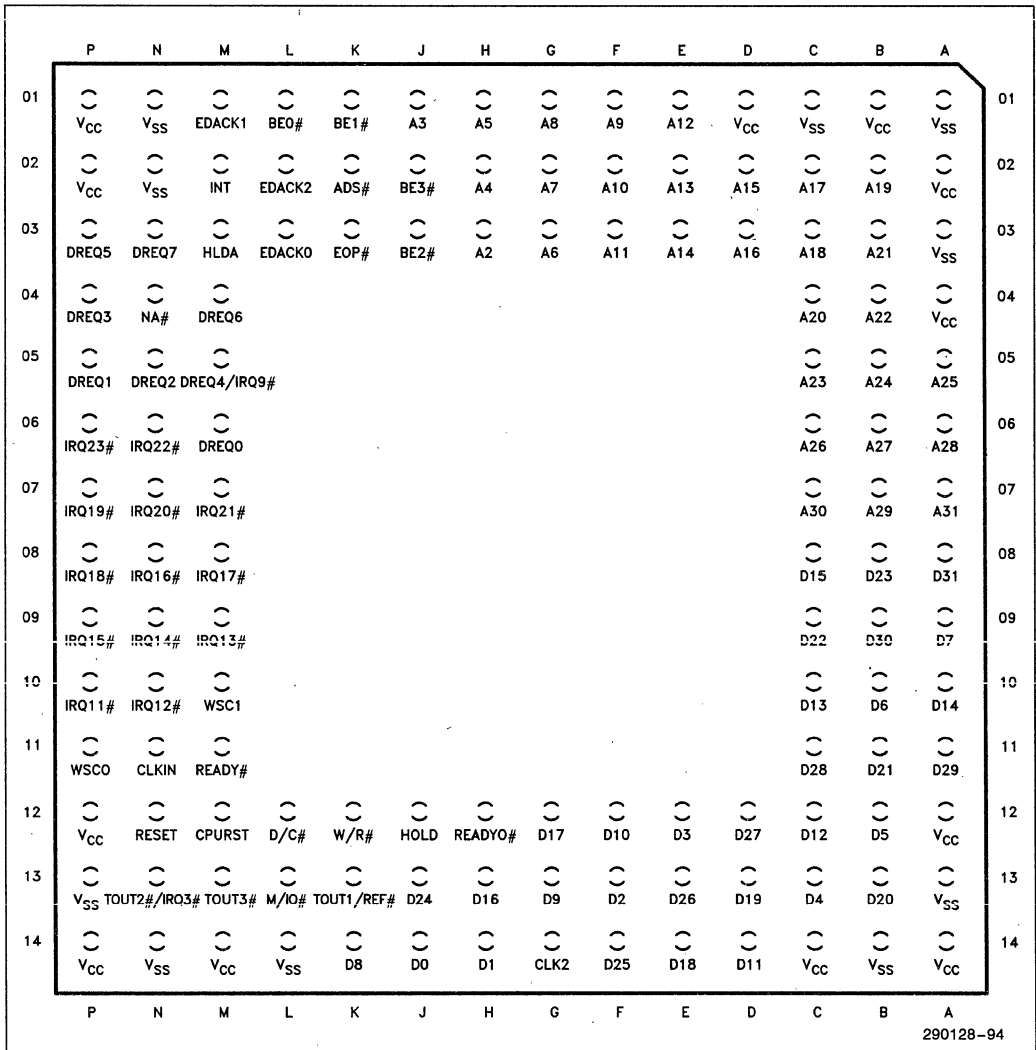


Figure 12.1. 82380 PGA Pinout—View from TOP side

12.2 Pin Assignment

The 82380 pinout as viewed from the top side of the component is shown in Figure 12.1. Its pinout as viewed from the pin side of the component is shown in Figure 12.2.

V_{CC} and GND connections must be made to multiple V_{CC} and V_{SS} (GND) pins. Each V_{CC} and V_{SS} MUST be connected to the appropriate voltage level. The circuit board should include V_{CC} and GND planes for power distribution and all V_{CC} pins must be connected to the appropriate plane.

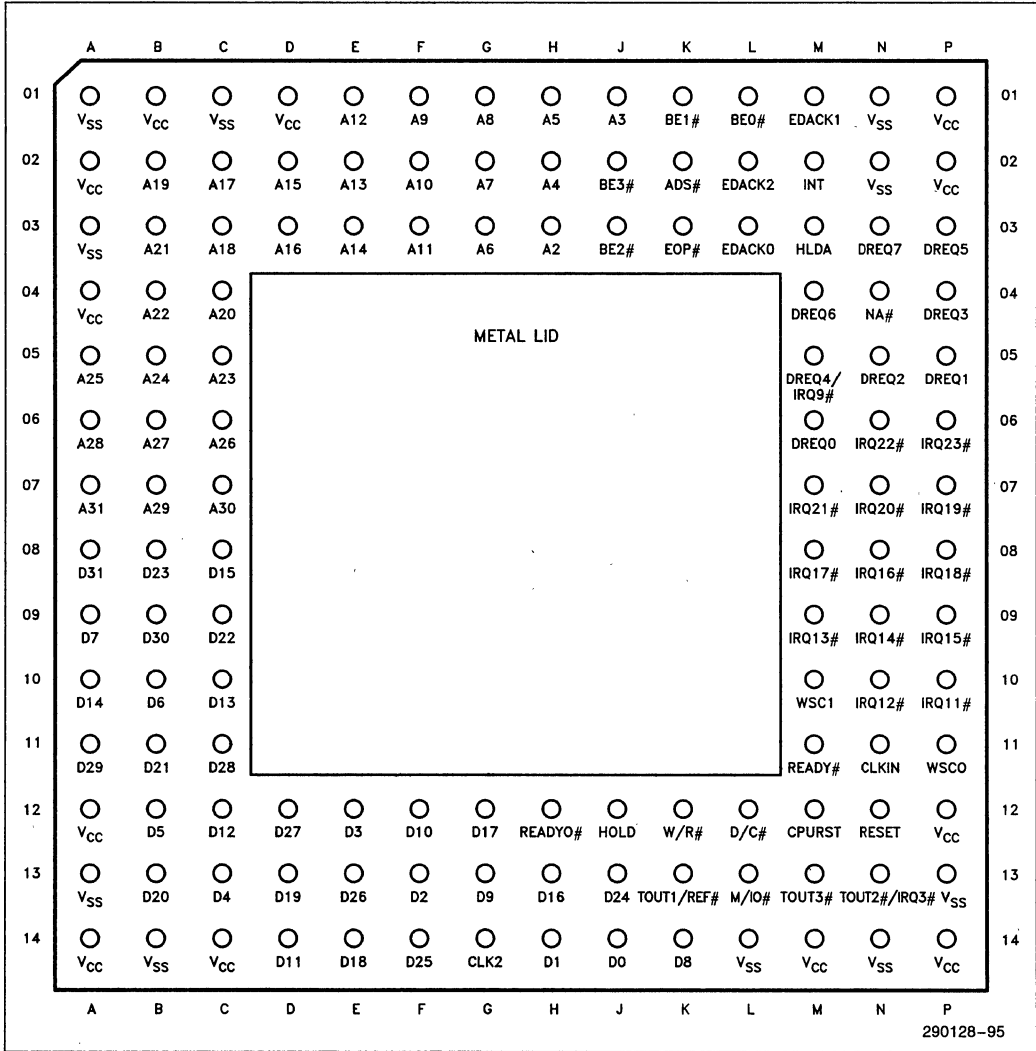


Figure 12.2. 82380 PGA Pinout—View from PIN side

Table 12-1. 82380 PGA Pinout—Functional Grouping

Pin/Signal		Pin/Signal		Pin/Signal		Pin/Signal	
A7	A31	A8	D31	P12	V _{CC}	L14	V _{SS}
C7	A30	B9	D30	M14	V _{CC}	A1	V _{SS}
B7	A29	A11	D29	P1	V _{CC}	P13	V _{SS}
A6	A28	C11	D28	P2	V _{CC}	N1	V _{SS}
B6	A27	D12	D27	P14	V _{CC}	N2	V _{SS}
C6	A26	E13	D26	D1	V _{CC}	C1	V _{SS}
A5	A25	F14	D25	C14	V _{CC}	A3	V _{SS}
B5	A24	J13	D24	B1	V _{CC}	B14	V _{SS}
C5	A23	B8	D23	A2	V _{CC}	A13	V _{SS}
B4	A22	C9	D22	A4	V _{CC}	N14	V _{SS}
B3	A21	B11	D21	A12	V _{CC}		
C4	A20	B13	D20	A14	V _{CC}	P6	IRQ23#
B2	A19	D13	D19			N6	IRQ22#
C3	A18	E14	D18	G14	CLK2	M7	IRQ21#
C2	A17	G12	D17	L12	D/C#	N7	IRQ20#
D3	A16	H13	D16	K12	W/R#	P7	IRQ19#
D2	A15	C8	D15	L13	M/IO#	P8	IRQ18#
E3	A14	A10	D14	K2	ADS#	M8	IRQ17#
E2	A13	C10	D13	N4	NA#	N8	IRQ16#
E1	A12	C12	D12	J12	HOLD	P9	IRQ15#
F3	A11	D14	D11	M3	HLDA	N9	IRQ14#
F2	A10	F12	D10	M6	DREQ0	M9	IRQ13#
F1	A9	G13	D9	P5	DREQ1	N10	IRQ12#
G1	A8	K14	D8	N5	DREQ2	P10	IRQ11#
G2	A7	A9	D7	P4	DREQ3	M2	INT
G3	A6	B10	D6	M5	DREQ4/IRQ9#		
H1	A5	B12	D5	P3	DREQ5	N11	CLKIN
H2	A4	C13	D4	M4	DREQ6	K13	TOUT1/REF#
J1	A3	E12	D3	N3	DREQ7	N13	TOUT2#/IRQ3#
H3	A2	F13	D2			M13	TOUT3#
J2	BE3#	H14	D1	K3	EOP#	M11	READY#
J3	BE2#	J14	D0	L3	EDACK0	H12	READYO#
K1	BE1#			M1	EDACK1	P11	WSC0
L1	BE0#	N12	RESET	L2	EDACK2	M10	WSC1
		M12	CPURST				

12.3 Package Dimensions and Mounting

The 82380 package is a 132-pin ceramic Pin Grid Array (PGA). The pins are arranged 0.100 inch (2.54 mm) center-to-center, in a 14 x 14 matrix, three rows around.

A wide variety of available sockets allow low insertion force or zero insertion force mountings, and a choice of terminals such as soldertail, surface mount, or wire wrap. Several applicable sockets are listed in Figure 12-4.

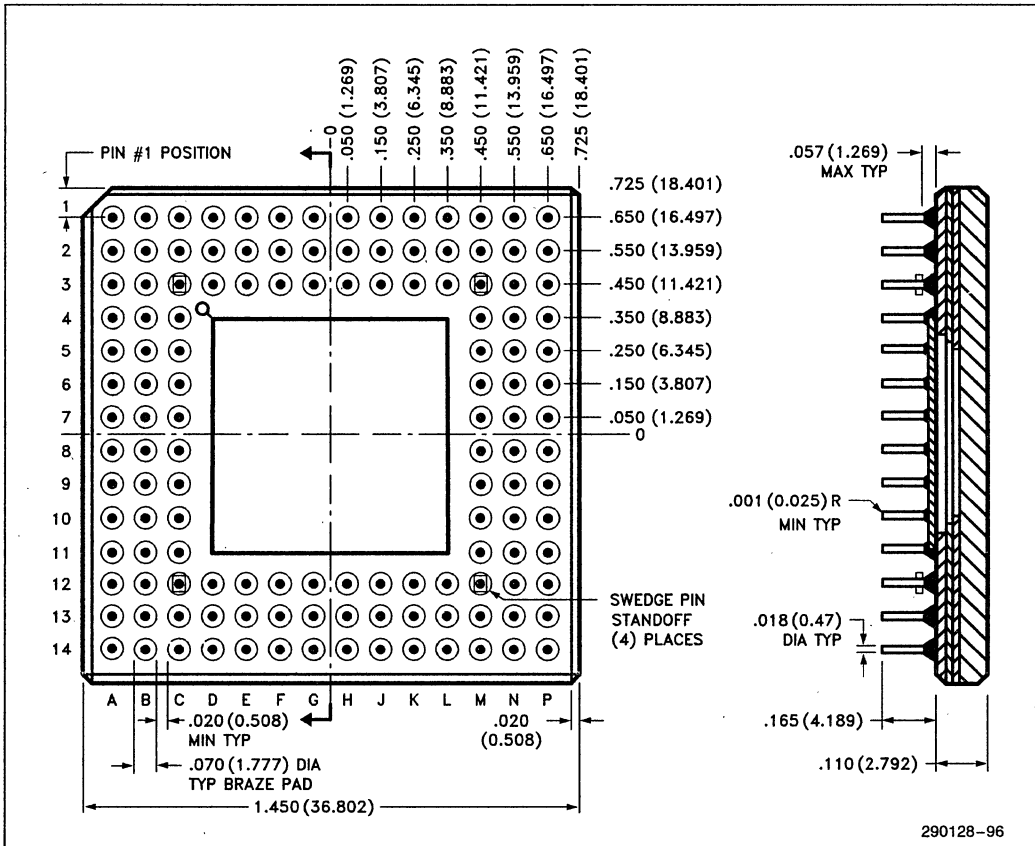
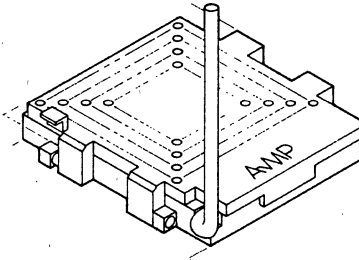
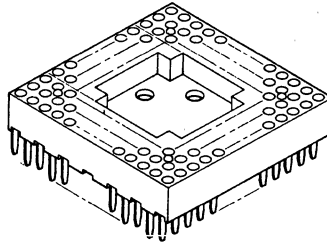


Figure 12.3. 132-Pin Ceramic PGA Package Dimensions

290128-96

- Low insertion force (LIF) soldertail 55274-1
 - Amp tests indicate 50% reduction in insertion force compared to machined sockets
- Other socket options
- Zero insertion force (ZIF) soldertail 55583-1
 - Zero insertion force (ZIF) Burn-in version 55573-2

Amp Incorporated
 (Harrisburg, PA 17105 U.S.A.)
 Phone 717-564-0100



290128-97

Cam handle locks in low profile position when substrate is installed
 (handle UP for open and DOWN for closed positions)

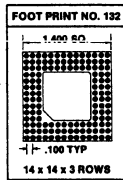
courtesy Amp Incorporated

Peel-A-Way™ Mylar and Kapton
 Socket Terminal Carriers

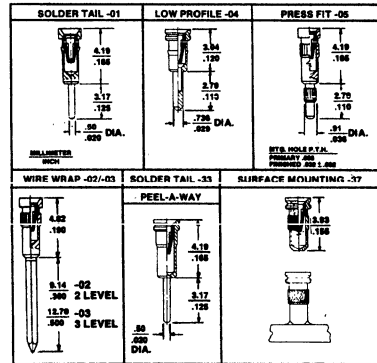
- Low insertion force surface mount CS132-37TG
- Low insertion force soldertail CS132-01TG
- Low insertion force wire-wrap CS132-02TG (two level)
 CS132-03TG (three-level)
- Low insertion force press-fit CS132-05TG

Advanced Interconnections
 (5 Division Street
 Warwick, RI 02818 U.S.A.)
 Phone 401-885-0485)

Peel-A-Way Carrier No. 132;
 Kapton Carrier is KS132
 Mylar Carrier is MS132
 Molded Plastic Body KS132
 is shown below:



290128-98



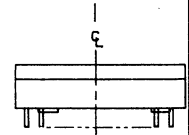
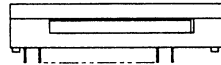
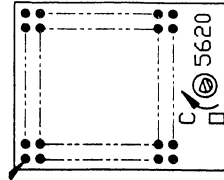
290128-99

courtesy Advanced Interconnections
 (Peel-A-Way Terminal Carriers
 U.S. Patent No. 4442938)

Figure 12-4. Several Socket Options for 132-pin PGA

- Low insertion force socket soldertail (for production use)
2XX-6576-00-3308 (new style)
2XX-6003-00-3302 (older style)
- Zero insertion force soldertail (for test and burn-in use)
2XX-6568-00-3302

Textool Products
Electronic Products Division/3M
 (1410 West Pioneer Drive
 Irving, Texas 75601 U.S.A.
 Phone 214-259-2676)



courtesy Textool Products/3M

290128-A0

Figure 12-4. Several Socket Options for 132-pin PGA (Continued)

12.4 Package Thermal Specification

The 82380 is specified for operation when case temperature is within the range of 0°C – 85°C. The case temperature may be measured in any environment,

to determine whether the 82380 is within the specified operating range.

The PGA case temperature should be measured at the center of the top surface opposite the pins, as in Figure 12.5.

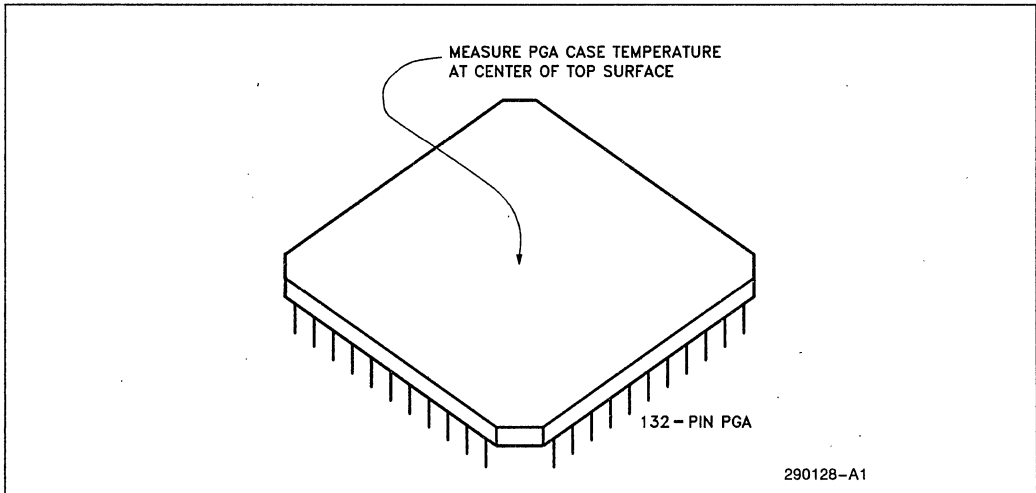


Figure 12.5. Measuring 82380 PGA Case Temperature

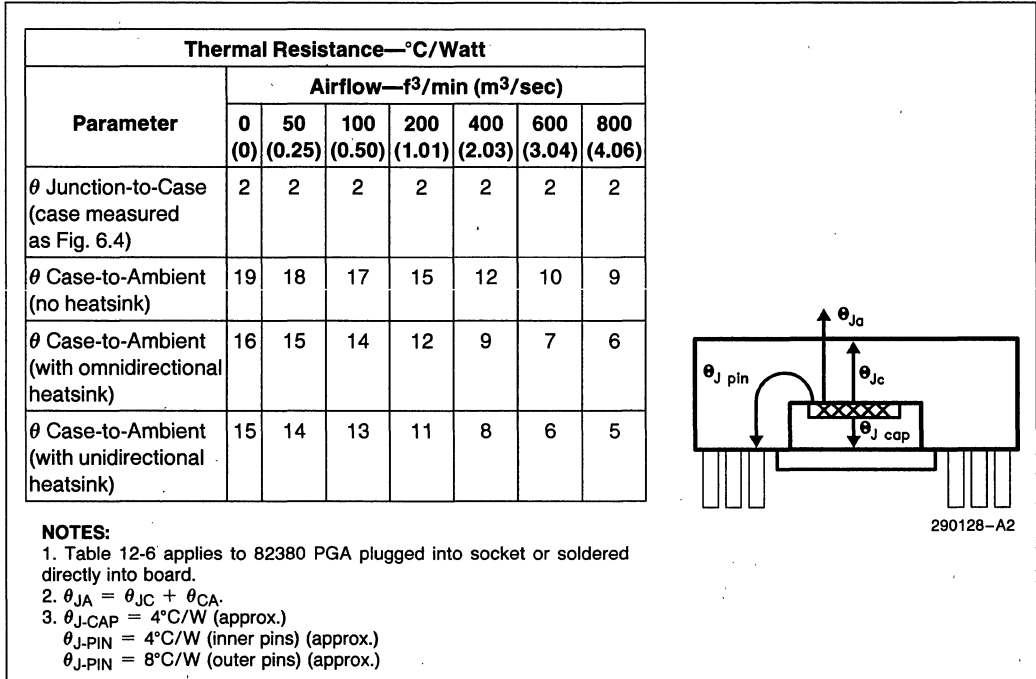


Figure 12-6. 82380 PGA Package Typical Thermal Characteristics

13.0 ELECTRICAL DATA

13.1 Power and Grounding

The large number of output buffers (address, data and control) can cause power surges as multiple output buffers drive new signal levels simultaneously. The 22 V_{CC} and V_{SS} pins of the 82380 each feed separate functional units to minimize switching induced noise effects. All V_{CC} pins of the 82380 must be connected on the circuit board.

13.2 Power Decoupling

Liberal decoupling capacitance should be placed close to the 82380. The 82380 driving its 32-bit parallel address and data buses at high frequencies can cause transient power surges when driving large capacitive loads. Low inductance capacitors and inter-

connects are recommended for the best reliability at high frequencies. Low inductance capacitors are available specifically for Pin Grid Array packages.

13.3 Unused Pin Recommendations

For reliable operation, ALWAYS connect unused inputs to a valid logic level. As is the case with most other CMOS processes, a floating input will increase the current consumption of the component and give an indeterminate state to the component.

13.4 ICE-386 Support

The 82380 specifications provide sufficient drive capability to support the ICE386. On the pins that are generally shared between the 80386 and the 82380, the additional loading represented by the ICE386 was allowed for in the design of the 82380.

13.5 Maximum Ratings

Storage Temperature -65°C to +150°C
 Case temperature Under Bias ... -65°C to +110°C
 Supply Voltage with Respect
 to V_{SS} -0.5V to +6.5V
 Voltage on any other Pin -0.5V to $V_{CC} + 0.5V$

NOTE:

Stress above those listed above may cause permanent damage to the device. This is a stress rating

only and functional operation at these or any other conditions above those listed in the operational sections of this specification is not implied.

Exposure to absolute maximum rating conditions for extended periods may affect device reliability. Although the 82380 contains protective circuitry to reset damage from static electric discharges, always take precautions against high static voltages or electric fields.

13.6 D.C. Specifications

$T_{CASE} = 0^{\circ}C$ to $85^{\circ}C$; $V_{CC} = 5V \pm 5\%$; $V_{SS} = 0V$.

Table 13-1.

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	-0.3	0.8	V	(Note 1)
V_{IH}	Input High Voltage	2.0	$V_{CC} + 0.3$	V	
V_{ILC}	CLK2 Input Low Voltage	-0.3	0.8		(Note 1)
V_{IHC}	CLK2 Input High Voltage	$V_{CC} - 0.8$	$V_{CC} + 0.3$	V	
V_{OL}	Output Low Voltage $I_{OL} = 4$ mA: A2-A31, D0-D31 $I_{OL} = 5$ mA: All Others		0.45	V	
			0.45	V	
V_{OH}	Output High Voltage $I_{OH} = -1$ mA: A2-A31, D0-D31 $I_{OH} = -0.9$ mA: All Others	2.4		V	
		2.4		V	
I_{LI}	Input Leakage Current for all ins except: IRQ11 # - IRQ23 #, TOUT2/IRQ3 #, EOP #, DREQ4		± 15	μA	$0V < V_{IN} < V_{CC}$
I_{LI1}	Input Leakage Current for pins: IRQ11 # - IRQ23 #, TOUT2 # / IRQ3 #, EOP #, DREQ4	10	-300	μA	$0V < V_{IN} < V_{CC}$ (Note 3)
I_{LO}	Output Leakage Current		± 15	μA	$0.45 < V_{OUT} < V_{CC}$
I_{CC}	Supply Current		300	mA	CLK2 = 32 MHz = 40 MHz (Note 4)
			325	mA	
(CAP)	Capacitance (Input/IO)		12	pF	$f_c = 1$ MHz (Note 2)
CCLK	CLK2 Capacitance		20	pF	$f_c = 1$ MHz (Note 2)

NOTES:

1. Minimum value is not 100% tested.
2. Sampled only.
3. These pins have internal pullups on them.
4. I_{CC} is specified with inputs driven to CMOS levels. I_{CC} may be higher if driven to TTL levels.

13.6 D.C. Specifications (Continued)

 $T_{CASE} = 0^{\circ}C$ to $85^{\circ}C$; $V_{CC} = 5V \pm 5\%$; $V_{SS} = 0V$.

Table 13-2. 82380-25 D.C. Specifications

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	-0.3	0.8	V	(Note 1)
V_{IH}	Input High Voltage	2.0	$V_{CC} + 0.3$	V	
V_{ILC}	CLK2 Input Low Voltage	-0.3	0.8	V	(Note 1)
V_{IHC}	CLK2 Input High Voltage	$V_{CC} - 0.8$	$V_{CC} + 0.3$	V	
V_{OL}	Output Low Voltage $I_{OL} = 4$ mA: A ₂ -A ₃₁ , D ₀ -D ₃₁ $I_{OL} = 5$ mA: All Others		0.45	V	
			0.45	V	
V_{OH}	Output High Voltage $I_{OH} = -1$ mA: A ₂ -A ₃₁ , D ₀ -D ₃₁ $I_{OH} = -0.9$ mA: All Others	2.4		V	
		2.4		V	
I_{LI}	Input Leakage Current All Inputs except: IRQ11# - IRQ23#, EOP#, TOUT2/IRQ3#, DREQ4		± 15	μA	
I_{LI1}	Input Leakage Current Inputs: IRQ11# -IRQ23#, EOP#, TOUT2/IRQ3#, DREQ4	10	-300	μA	$0 < V_{IN} < V_{CC}$ (Note 3)
I_{LO}	Output Leakage Current		± 15	μA	$0 < V_{IN} < V_{CC}$
I_{CC}	Supply Current (CLK2 = 50 MHz)		375	mA	(Note 4)
C_I	Input Capacitance		12	pF	(Note 2)
C_{CLK}	CLK2 Input Capacitance		20	pF	(Note 2)

NOTES:

- Minimum value is not 100% tested.
- $f_c = 1$ MHz; Sampled only.
- These pins have weak internal pullups. They should not be left floating.
- I_{CC} is specified with inputs driven to CMOS levels, and outputs driving CMOS loads. I_{CC} may be higher if inputs are driven to TTL levels, or if outputs are driving TTL loads.

13.7 A.C. Specifications

The A.C. specifications given in the following tables consist of output delays and input setup requirements. The A.C. diagram's purpose is to illustrate the clock edges from which the timing parameters are measured. The reader should not infer any other timing relationships from them. For specific information on timing relationships between signals, refer to the appropriate functional section.

A.C. spec measurement is defined in Figure 13.1. Inputs must be driven to the levels shown when A.C. specifications are measured. 82380 output delays are specified with minimum and maximum limits, which are measured as shown. The minimum 82380 output delay times are hold times for external circuitry. 82380 input setup and hold times are specified as minimums and define the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 82380 operation.

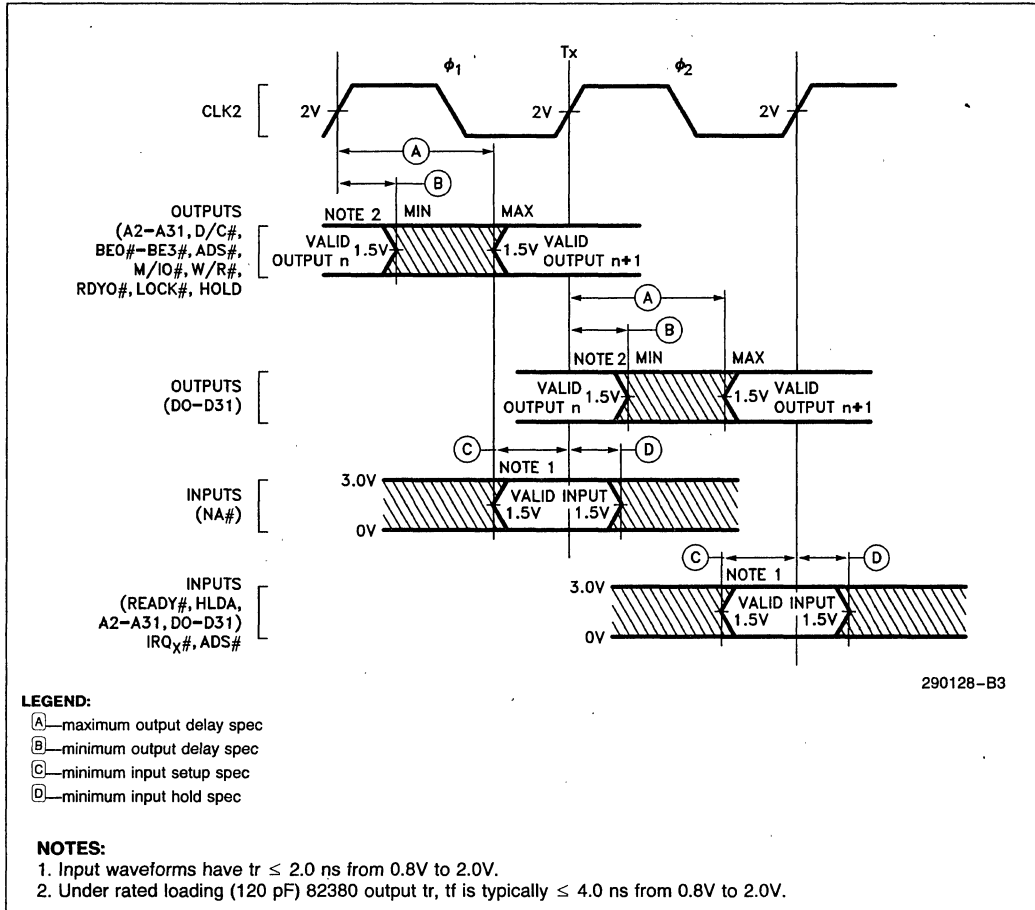


Figure 13-1. Drive Levels and Measurement Points for A.C. Specification

A.C. SPECIFICATION TABLES

 Functional Operating Range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$
Table 13-3. 82380 A.C. Characteristics

Symbol	Parameter	82380-16		82380-20		Notes
		Min	Max	Min	Max	
	Operating Frequency	4 MHz	16 MHz	4 MHz	20 MHz	Half CLK2 Frequency
t1	CLK2 Period	31 ns	125 ns	25 ns	125 ns	
t2a	CLK2 High Time	9		8		at 2.0V
t2b	CLK2 High Time	5		5		at $(V_{CC}-0.8)V$
t3a	CLK2 Low Time	9		8		at 2.0V
t3b	CLK2 Low Time	7		6		at 0.8V
t4	CLK2 Fall Time		8		8	$(V_{CC}-0.8)V$ to 0.8V
t5	CLK2 Rise Time		8		8	0.8V to $(V_{CC}-0.8)V$
t6	A (2-31), BE (0-3) #, EDACK (0-2) Valid Delay	4	36	4	30	CL = 120 pF (Note 1)
t7	Float Delay	4	40	4	32	
t8	A (2-31), BE (0-3) # Setup Time	6		6		
t9	Hold Time	4		4		
t10	W/R#, M/IO#, D/C#, Valid Delay	6	33	6	28	CL = 75 pF (Note 1)
t11	Float Delay	4	35	4	30	
t12	Setup Time	6		6		
t13	Hold Time	4		4		
t14	ADS# Valid Delay	6	33	6	28	CL = 75 pF
t15	Float Delay	4	35	4	30	
t16	Setup Time	21		15		
t17	Hold Time	4		4		
t18	Slave Mode— D(0-31) Read Valid Delay	3	46	4	46	CL = 120 pF (Note 1)
t19	Float Delay	6	35	6	29	
t20	Slave Mode— D(0-31) Write Setup Time			29		
t21	Hold Time	31		26		

A.C. SPECIFICATION TABLES (Continued)

 Functional Operating Range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$.

Table 13-3. 82380 A.C. Characteristics (Continued)

Symbol	Parameter	82380-16		82380-20		Notes
		Min	Max	Min	Max	
t22	Master Mode— D(0–31) Write Valid Delay	4	48	4	38	CL = 120 pF (Note 1)
t23	Float Delay	4	35	4	27	
t24	Master Mode— D(0–31) Read Setup Time	11		11		
t25	Hold Time	6		6		
t26	READY# Setup Time	21		12		
t27	Hold Time	4		4		
t28	WSC (0–1) Setup	6		6		
t29	Hold	21		21		
t31	RESET Setup Time	13		12		
t30	Hold Time	4		4		
t32	READYO# Valid Delay	4	31	4	28	CL = 25 pF
t33	CPU Reset From CLK2	2	18	2	16	CL = 50 pF
t34	HOLD Valid Delay	5	33	5	30	CL = 100 pF
t35	HLDA Setup Time	21		17		
t36	Hold Time	6		6		
t37a	EOP# Setup Time	21		17		Synch. EOP
t38a	EOP# Hold Time	4		4		
t37b	EOP# Setup Time	11		11		Asynch. EOP
t38b	EOP# Hold Time	11		11		
t39	EOP# Valid Delay	5	38	5	30	CL = 100 pF ('1'-'>'0')
t40	EOP# Float Delay	5	40	5	32	
t41a	DREQ Setup Time	21		19		Synchronous DREQ
t42a	Hold Time	4		4		
t41b	DREQ Setup Time	11		11		Asynchronous DREQ
t42b	Hold Time	11		11		
t43	INT Valid Delay		500		500	From IRQ Input CL = 75 pF
t44	NA# Setup Time	11		10		
t45	Hold Time	15		15		

A.C. SPECIFICATION TABLES (Continued)

Functional Operating Range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$.

Table 13-3. 82380 A.C. Characteristics (Continued)

Symbol	Parameter	82380-16		82380-20		Notes
		Min	Max	Min	Max	
t46	CLKIN Frequency	0 MHz	10 MHz	0 MHz	10 MHz	
t47	CLKIN High Time	30		30		At 1.5V
t48	CLKIN Low Time	50		50		At 1.5V
t49	CLKIN Rise Time		10		10	0.8V to 2.0V
t50	CLKIN Fall Time		10		10	2.0V to 0.8V
t51	TOUT1/REF# Valid	4	36	4	30	From CLK2, CL = 25 pF
t52	TOUT1/REF# Valid	3	93	3	93	From CLKIN, CL = 120 pF
t53	TOUT2# Valid Delay	3	93	3	93	From CLKIN, CL = 120 pF (Falling Edge Only)
t54	TOUT2# Float Delay	3	40	3	40	From CLKIN (Note 1)
t55	TOUT3# Valid Delay	3	93	3	93	From CLKIN, CL = 120 pF

NOTE:

1. Float condition occurs when the maximum output current becomes less than ILO in magnitude. Float delay is not tested. For testing purposes, the float condition occurs when the dynamic output driven voltage changes with current loads.

Functional Operating Range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$.

A.C. timings are tested at 1.5V thresholds; except as noted.

Table 13-4. 82380-25 A.C. Characteristics

Symbol	Parameter	82380-25		Unit	Notes
		Min	Max		
	Operating Frequency $1/(t1a \times 2)$	4	25	MHz	
t1	CLK2 Period	20	125	ns	
t2a	CLK2 High Time	7		ns	at 2.0V
t2b	CLK2 High Time	4		ns	at 3.7V
t3a	CLK2 Low Time	7		ns	at 2.0V
t3b	CLK2 Low Time	4		ns	at 0.8V
t4	CLK2 Fall Time		7	ns	3.7V to 0.8V
t5	CLK2 Rise Time		7	ns	0.8V to 3.7V
t6	A2-A31, BE0#-BE3# EDACK0-EDACK3 Valid Delay	4	20	ns	50 pF Load
t7	A2-A31, BE0#-BE3# EDACK0-EDACK3 Float Delay	4	27	ns	50 pF Load
t8	A2-A31, BE0#-BE3# Setup Time	6		ns	
t9	A2-A31, BE0#-BE3# Hold Time	4		ns	
t10	W/R#, M/IO#, D/C# Valid Delay	4	20	ns	50 pF Load
t11	W/R#, M/IO#, D/C# Float Delay	4	29	ns	50 pF Load

A.C. SPECIFICATION TABLES (Continued)

 Functional Operating Range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$.

A.C. timings are tested at 1.5V thresholds; except as noted.

Table 13-4. 82380-25 A.C. Characteristics (Continued)

Symbol	Parameter	82380-25		Unit	Notes
		Min	Max		
t12	W/R#, M/IO#, D/C# Setup Time	6		ns	
t13	W/R#, M/IO#, D/C# Hold Time	4		ns	
t14	ADS# Valid Delay	4	19	ns	50 pF Load
t15	ADS# Float Delay	4	29	ns	50 pF Load
t16	ADS# Setup Time	12		ns	
t17	ADS# Hold Time	4		ns	
t18	Slave Mode D0–D31 Read Valid	4	31	ns	50 pF Load
t19	Slave Mode D0–D31 Read Float	6	21	ns	50 pF Load
t20	Slave Mode D0–D31 Write Setup	20		ns	
t21	Slave Mode D0–D31 Write Hold	20		ns	
t22	Master Mode D0–D31 Write Valid	8	27	ns	50 pF Load
t23	Master Mode D0–D31 Write Float	4	19	ns	50 pF Load
t24	Master Mode D0–D31 Read Setup	7		ns	
t25	Master Mode D0–D31 Read Hold	4		ns	
t26	READY# Setup Time	9		ns	
t27	READY# Hold Time	4		ns	
t28	WSC0–WSC1 Setup Time	6		ns	
t29	WSC0–WSC1 Hold Time	15		ns	
t30	RESET Hold Time	4		ns	
t31	RESET Setup Time	9		ns	
t32	READYO# Valid Delay	3	21	ns	25 pF Load
t33	CPURST Valid Delay	2	14	ns	50 pF Load
t34	HOLD Valid Delay	4	22	ns	50 pF Load
t35	HLDA Setup Time	17		ns	
t36	HLDA Hold Time	4		ns	
t37a	EOP# Setup (Synchronous)	13		ns	
t38a	EOP# Hold (Synchronous)	4		ns	
t37b	EOP# Setup (Asynchronous)	10		ns	
t38b	EOP# Hold (Asynchronous)	10		ns	
t39	EOP# Valid Delay	4	21	ns	50 pF Load
t40	EOP# Float Delay	4	21	ns	50 pF Load
t41a	DREQ Setup (Synchronous)	17		ns	
t42a	DREQ Hold (Synchronous)	4		ns	
t41b	DREQ Setup (Asynchronous)	10		ns	
t42b	DREQ Hold (Asynchronous)	10		ns	
t43	INT Valid Delay from IRQn		500	ns	50 pF Load

A.C. SPECIFICATION TABLES (Continued)

Functional Operating Range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0^{\circ}C$ to $+85^{\circ}C$.

A.C. timings are tested at 1.5V thresholds; except as noted.

Table 13-4. 82380-25 A.C. Characteristics (Continued)

Symbol	Parameter	82380-25		Unit	Notes
		Min	Max		
t44	NA # Setup Time	7		ns	
t45	NA # Hold Time	8		ns	
t46	CLKIN Frequency	0	10	MHz	
t47	CLKIN High Time	30		ns	2.0V
t48	CLKIN Low Time	50		ns	0.8V
t49	CLKIN Rise Time		10	ns	0.8V to 3.7V
t50	CLKIN Fall Time		10	ns	3.7V to 0.8V
t51	TOUT1/REF # Valid Delay from CLK2 (Refresh)	4	20	ns	50 pF Load
t52	TOUT1/REF # Valid Delay from CLKIN (Timer)	3	90	ns	50 pF Load
t53	TOUT2 # Valid Delay (Falling Edge Only)	3	90	ns	50 pF Load
t54	TOUT2 # Float Delay	3	37	ns	50 pF Load
t55	TOUT3 # Valid Delay	3	90	ns	50 pF Load

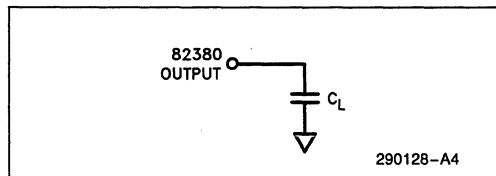


Figure 13-2. A.C. Test Load

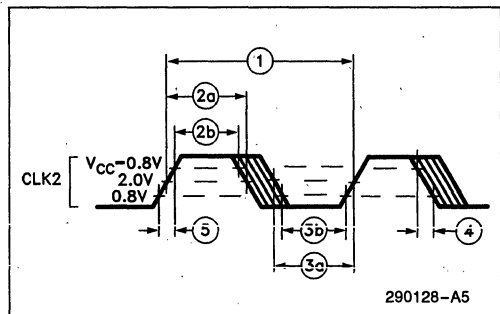
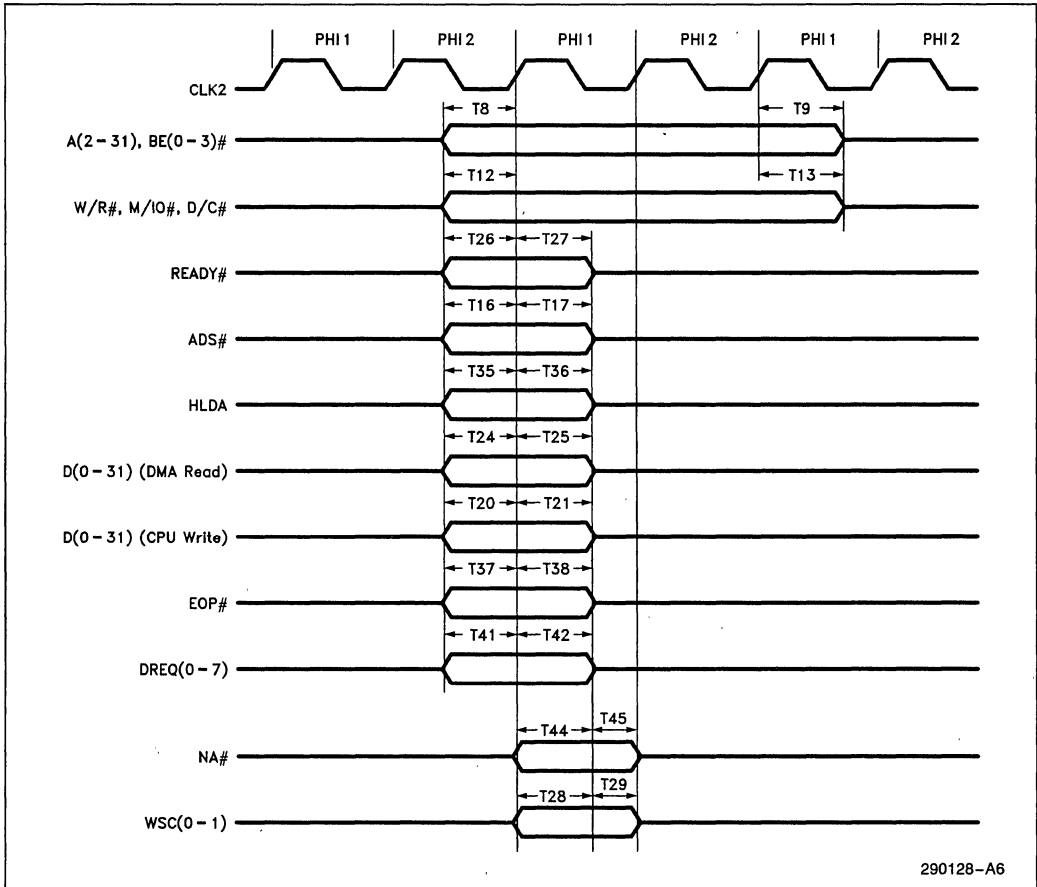
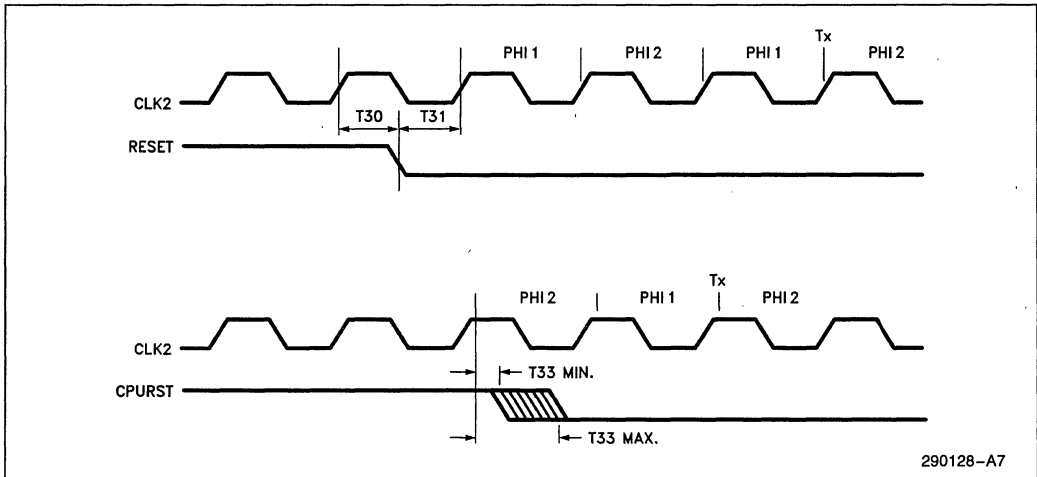


Figure 13-3. CLK2 Timing



290128-A6

Figure 13-4. Input Setup and Hold Timing



290128-A7

Figure 13-5. Reset Timing

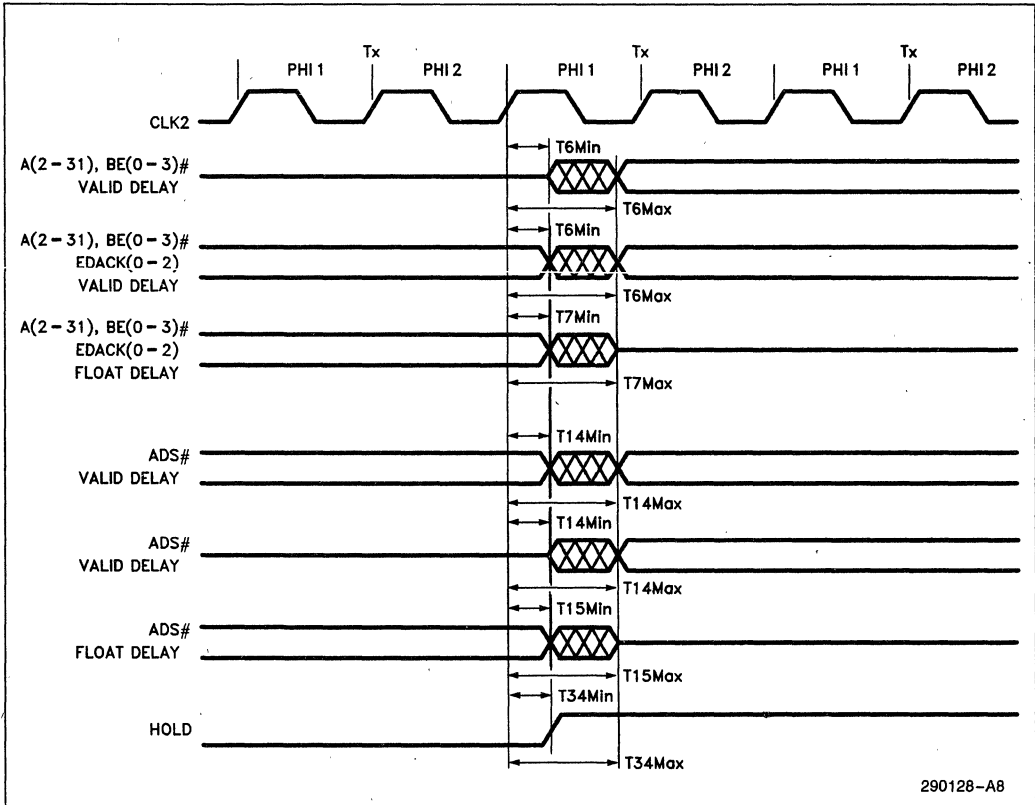


Figure 13-6. Address Output Delays

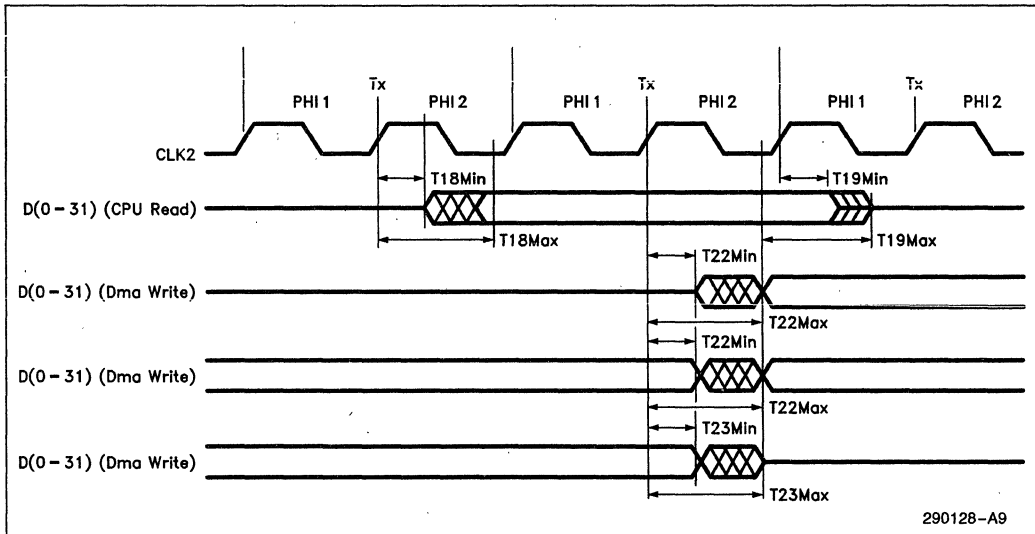


Figure 13-7. Data Bus Output Delays

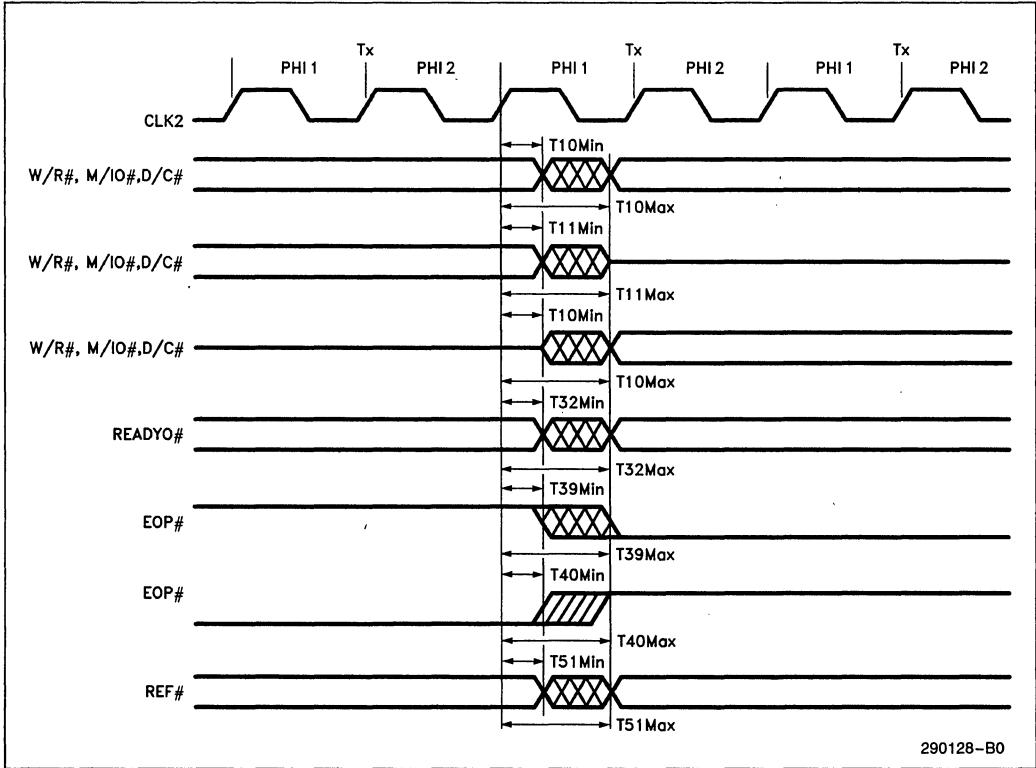


Figure 13-8. Control Output Delays

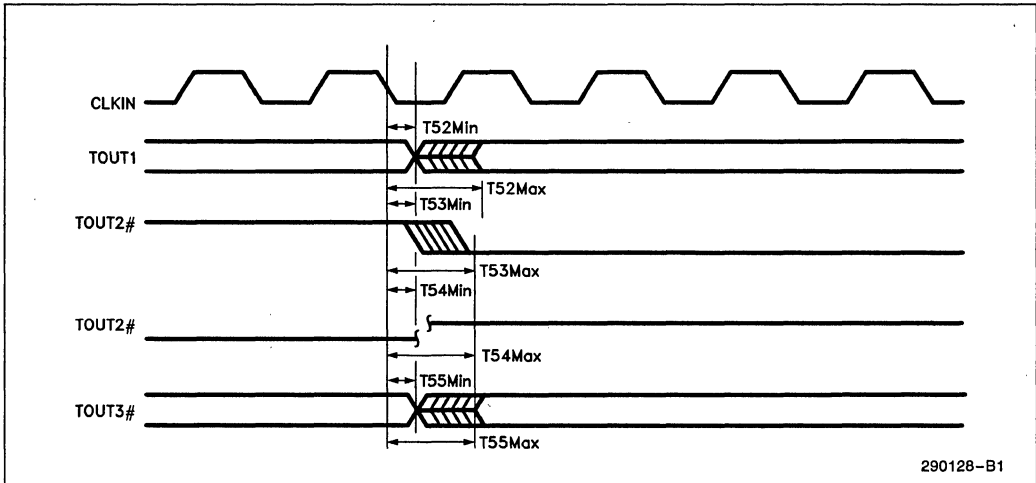


Figure 13-9. Timer Output Delays

APPENDIX A

Ports Listed by Address

Port Address (HEX)	Description
00	Read/Write DMA Channel 0 Target Address, A0–A15
01	Read/Write DMA Channel 0 Byte Count, B0–B15
02	Read/Write DMA Channel 1 Target Address, A0–A15
03	Read/Write DMA Channel 1 Byte Count, B0–B15
04	Read/Write DMA Channel 2 Target Address, A0–A15
05	Read/Write DMA Channel 2 Byte Count, B0–B15
06	Read/Write DMA Channel 3 Target Address, A0–A15
07	Read/Write DMA Channel 3 Byte Count, B0–B15
08	Read/Write DMA Channel 0–3 Status/Command I Register
09	Read/Write DMA Channel 0–3 Software Request Register
0A	Write DMA Channel 0–3 Set-Reset Mask Register
0B	Write DMA Channel 0–3 Mode Register I
0C	Write Clear Byte-Pointer FF
0D	Write DMA Master-Clear
0E	Write DMA Channel 0–3 Clear Mask Register
0F	Read/Write DMA Channel 0–3 Mask Register
10	Read/Write DMA Channel 0 Target Address, A24–A31
11	Read/Write DMA Channel 0 Byte Count, B16–B23
12	Read/Write DMA Channel 1 Target Address, A24–A31
13	Read/Write DMA Channel 1 Byte Count, B16–B23
14	Read/Write DMA Channel 2 Target Address, A24–A31
15	Read/Write DMA Channel 2 Byte Count, B16–B23
16	Read/Write DMA Channel 3 Target Address, A24–A31
17	Read/Write DMA Channel 3 Byte Count, B16–B23
18	Write DMA Channel 0–3 Bus Size Register
19	Read/Write DMA Channel 0–3 Chaining Register
1A	Write DMA Channel 0–3 Command Register II
1B	Write DMA Channel 0–3 Mode Register II
1C	Read/Write Refresh Control Register
1E	Reset Software Request Interrupt
20	Write Bank B ICW1, OCW2, or OCW3
	Read Bank B Poll, Interrupt Request or In-Service Status Register
21	Write Bank B ICW2, ICW3, ICW4 or OCW1
	Read Bank B Interrupt Mask Register
22	Read Bank B ICW2
28	Read/Write IRQ8 Vector Register
29	Read/Write IRQ9 Vector Register
2A	Reserved
2B	Read/Write IRQ11 Vector Register
2C	Read/Write IRQ12 Vector Register
2D	Read/Write IRQ13 Vector Register
2E	Read/Write IRQ14 Vector Register
2F	Read/Write IRQ15 Vector Register

APPENDIX A—Ports Listed by Address (Continued)

Port Address (HEX)	Description
30	Write Bank A ICW1, OCW2 or OCW3 Read Bank A Poll, Interrupt Request or In-Service Status Register
31	Write Bank A ICW2, ICW3, ICW4 or OCW1 Read Bank A Interrupt Mask Register
32	Read Bank A ICW2
38	Read/Write IRQ0 Vector Register
39	Read/Write IRQ1 Vector Register
3A	Read/Write IRQ1.5 Vector Register
3B	Read/Write IRQ3 Vector Register
3C	Read/Write IRQ4 Vector Register
3D	Reserved
3E	Reserved
3F	Read/Write IRQ7 Vector Register
40	Read/Write Counter 0 Register
41	Read/Write Counter 1 Register
42	Read/Write Counter 2 Register
43	Write Control Word Register I—Counter 0, 1, 2
44	Read/Write Counter 3 Register
45	Reserved
46	Reserved
47	Write Word Register II—Counter 3
61	Write Internal Control Port
64	Write CPU Reset Register (Data-1111XXX0H)
72	Read/Write Wait State Register 0
73	Read/Write Wait State Register 1
74	Read/Write Wait State Register 2
75	Read/Write Refresh Wait State Register
76	Reserved
77	Reserved
7D	Reserved
7E	Reserved
7F	Read/Write Relocation Register
80	Read/Write Internal Diagnostic Port 0
81	Read/Write DMA Channel 2 Target Address, A16–A23
82	Read/Write DMA Channel 3 Target Address, A16–A23
83	Read/Write DMA Channel 1 Target Address, A16–A23
87	Read/Write DMA Channel 0 Target Address, A16–A23
88	Read/Write Internal Diagnostic Port 1
89	Read/Write DMA Channel 6 Target Address, A16–A23
8A	Read/Write DMA Channel 7 Target Address, A16–A23
8B	Read/Write DMA Channel 5 Target Address, A16–A23
8F	Read/Write DMA Channel 4 Target Address, A16–A23

APPENDIX A—Ports Listed by Address (Continued)

Port Address (HEX)	Description
90	Read/Write DMA Channel 0 Requester Address, A0–A15
91	Read/Write DMA Channel 0 Requester Address, A16–A31
92	Read/Write DMA Channel 1 Requester Address, A0–A15
93	Read/Write DMA Channel 1 Requester Address, A16–A31
94	Read/Write DMA Channel 2 Requester Address, A0–A15
95	Read/Write DMA Channel 2 Requester Address, A16–A31
96	Read/Write DMA Channel 3 Requester Address, A0–A15
97	Read/Write DMA Channel 3 Requester Address, A16–A31
98	Read/Write DMA Channel 4 Requester Address, A0–A15
99	Read/Write DMA Channel 4 Requester Address, A16–A31
9A	Read/Write DMA Channel 5 Requester Address, A0–A15
9B	Read/Write DMA Channel 5 Requester Address, A16–A31
9C	Read/Write DMA Channel 6 Requester Address, A0–A15
9D	Read/Write DMA Channel 6 Requester Address, A16–A31
9E	Read/Write DMA Channel 7 Requester Address, A0–A15
9F	Read/Write DMA Channel 7 Requester Address, A16–A31
A0	Write Bank C ICW1, OCW2 or OCW3 Read Bank C Poll, Interrupt Request or In-Service Status Register
A1	Write Bank C ICW2, ICW3, ICW4 or OCW1 Read Bank C Interrupt Mask Register
A2	Read Bank C ICW2
A8	Read/Write IRQ16 Vector Register
A9	Read/Write IRQ17 Vector Register
AA	Read/Write IRQ18 Vector Register
AB	Read/Write IRQ19 Vector Register
AC	Read/Write IRQ20 Vector Register
AD	Read/Write IRQ21 Vector Register
AE	Read/Write IRQ22 Vector Register
AF	Read/Write IRQ23 Vector Register
C0	Read/Write DMA Channel 4 Target Address, A0–A15
C1	Read/Write DMA Channel 4 Byte Count, B0–B15
C2	Read/Write DMA Channel 5 Target Address, A0–A15
C3	Read/Write DMA Channel 5 Byte Count, B0–B15
C4	Read/Write DMA Channel 6 Target Address, A0–A15
C5	Read/Write DMA Channel 6 Byte Count, B0–B15
C6	Read/Write DMA Channel 7 Target Address, A0–A15
C7	Read/Write DMA Channel 7 Byte Count, B0–B15
C8	Read DMA Channel 4–7 Status/Command I Register
C9	Read/Write DMA Channel 4–7 Software Request Register
CA	Write DMA Channel 4–7 Set—Reset Mask Register
CB	Write DMA Channel 4–7 Mode Register I
CC	Reserved
CD	Reserved
CE	Write DMA Channel 4–7 Clear Mask Register
CF	Read/Write DMA Channel 4–7 Mask Register

APPENDIX A—Ports Listed by Address (Continued)

Port Address (HEX)	Description
D0	Read/Write DMA Channel 4 Target Address, A24–A31
D1	Read/Write DMA Channel 4 Byte Count, B16–B23
D2	Read/Write DMA Channel 5 Target Address, A24–A31
D3	Read/Write DMA Channel 5 Byte Count, B16–B23
D4	Read/Write DMA Channel 6 Target Address, A24–A31
D5	Read/Write DMA Channel 6 Byte Count, B16–B23
D6	Read/Write DMA Channel 7 Target Address, A24–A31
D7	Read/Write DMA Channel 7 Byte Count, B16–B23
D8	Write DMA Channel 4–7 Bus Size Register
D9	Read/Write DMA Channel 4–7 Chaining Register
DA	Write DMA Channel 4–7 Command Register II
DB	Write DMA Channel 4–7 Mode Register II

APPENDIX B

Ports Listed by Function

Port Address (HEX)	Description
DMA CONTROLLER	
0D	Write DMA Master-Clear
0C	Write DMA Clear Byte-Pointer FF
08	Read/Write DMA Channel 0–3 Status/Command I Register
C8	Read/Write DMA Channel 4–7 Status/Command I Register
1A	Write DMA Channel 0–3 Command Register II
DA	Write DMA Channel 4–7 Command Register II
0B	Write DMA Channel 0–3 Mode Register I
CB	Write DMA Channel 4–7 Mode Register I
1B	Write DMA Channel 0–3 Mode Register II
DB	Write DMA Channel 4–7 Mode Register II
09	Read/Write DMA Channel 0–3 Software Request Register
C9	Read/Write DMA Channel 4–7 Software Request Register
1E	Reset Software Request Interrupt
0E	Write DMA Channel 0–3 Clear Mask Register
CE	Write DMA Channel 4–7 Clear Mask Register
0F	Read/Write DMA Channel 0–3 Mask Register
CF	Read/Write DMA Channel 4–7 Mask Register
0A	Write DMA Channel 0–3 Set-Reset Mask Register
CA	Write DMA Channel 4–7 Set-Reset Mask Register
18	Write DMA Channel 0–3 Bus Size Register
D8	Write DMA Channel 4–7 Bus Size Register
i9	Read/Write DMA Channel 0–3 Chaining Register
D9	Read/Write DMA Channel 4–7 Chaining Register
00	Read/Write DMA Channel 0 Target Address, A0–A15
87	Read/Write DMA Channel 0 Target Address, A16–A23
10	Read/Write DMA Channel 0 Target Address, A24–A31
01	Read/Write DMA Channel 0 Byte Count, B0–B15
11	Read/Write DMA Channel 0 Byte Count, B16–B23
90	Read/Write DMA Channel 0 Requester Address, A0–A15
91	Read/Write DMA Channel 0 Requester Address, A16–A31
02	Read/Write DMA Channel 1 Target Address, A0–A15
83	Read/Write DMA Channel 1 Target Address, A16–A23
12	Read/Write DMA Channel 1 Target Address, A24–A31
03	Read/Write DMA Channel 1 Byte Count, B0–B15
13	Read/Write DMA Channel 1 Byte Count, B16–B23
92	Read/Write DMA Channel 1 Requester Address, A0–A15
93	Read/Write DMA Channel 1 Requester Address, A16–A31

APPENDIX B—Ports Listed by Function (Continued)

Port Address (HEX)	Description
DMA CONTROLLER	
04	Read/Write DMA Channel 2 Target Address, A0–A15
81	Read/Write DMA Channel 2 Target Address, A16–A23
14	Read/Write DMA Channel 2 Target Address, A24–A31
05	Read/Write DMA Channel 2 Byte Count, B0–B15
15	Read/Write DMA Channel 2 Byte Count, B16–B23
94	Read/Write DMA Channel 2 Requester Address, A0–A15
95	Read/Write DMA Channel 2 Requester Address, A16–A31
06	Read/Write DMA Channel 3 Target Address, A0–A15
82	Read/Write DMA Channel 3 Target Address, A16–A23
16	Read/Write DMA Channel 3 Target Address, A24–A31
07	Read/Write DMA Channel 3 Byte Count, B0–B15
17	Read/Write DMA Channel 3 Byte Count, B16–B23
96	Read/Write DMA Channel 3 Requester Address, A0–A15
97	Read/Write DMA Channel 3 Requester Address, A16–A31
C0	Read/Write DMA Channel 4 Target Address, A0–A15
8F	Read/Write DMA Channel 4 Target Address, A16–A23
D0	Read/Write DMA Channel 4 Target Address, A24–A31
C1	Read/Write DMA Channel 4 Byte Count, B0–B15
D1	Read/Write DMA Channel 4 Byte Count, B16–B23
98	Read/Write DMA Channel 4 Requester Address, A0–A15
99	Read/Write DMA Channel 4 Requester Address, A16–A31
C2	Read/Write DMA Channel 5 Target Address, A0–A15
8B	Read/Write DMA Channel 5 Target Address, A16–A23
D2	Read/Write DMA Channel 5 Target Address, A24–A31
C3	Read/Write DMA Channel 5 Byte Count, B0–B15
D3	Read/Write DMA Channel 5 Byte Count, B16–B23
9A	Read/Write DMA Channel 5 Requester Address, A0–A15
9B	Read/Write DMA Channel 5 Requester Address, A16–A31
C4	Read/Write DMA Channel 6 Target Address, A0–A15
89	Read/Write DMA Channel 6 Target Address, A16–A23
D4	Read/Write DMA Channel 6 Target Address, A24–A31
C5	Read/Write DMA Channel 6 Byte Count, B0–B15
D5	Read/Write DMA Channel 6 Byte Count, B16–B23
9C	Read/Write DMA Channel 6 Requester Address, A0–A15
9D	Read/Write DMA Channel 6 Requester Address, A16–A31
C6	Read/Write DMA Channel 7 Target Address, A0–A15
8A	Read/Write DMA Channel 7 Target Address, A16–A23
D6	Read/Write DMA Channel 7 Target Address, A24–A31
C7	Read/Write DMA Channel 7 Byte Count, B0–B15
D7	Read/Write DMA Channel 7 Byte Count, B16–B23
9E	Read/Write DMA Channel 7 Requester Address, A0–A15
9F	Read/Write DMA Channel 7 Requester Address, A16–A31

APPENDIX B—Ports Listed by Function (Continued)

Port Address (HEX)	Description
INTERRUPT CONTROLLER	
20	Write Bank B ICW1, OCW2, or OCW3 Read Bank B Poll, Interrupt Request or In-Service Status Register
21	Write Bank B ICW2, ICW3, ICW4 or OCW1 Read Bank B Interrupt Mask Register
22	Read Bank B ICW2
28	Read/Write IRQ8 Vector Register
29	Read/Write IRQ9 Vector Register
2A	Reserved
2B	Read/Write IRQ11 Vector Register
2C	Read/Write IRQ12 Vector Register
2D	Read/Write IRQ13 Vector Register
2E	Read/Write IRQ14 Vector Register
2F	Read/Write IRQ15 Vector Register
A0	Write Bank C ICW1, OCW2 or OCW3 Read Bank C Poll, Interrupt Request or In-Service Status Register
A1	Write Bank C ICW2, ICW3, ICW4 or OCW1 Read Bank C Interrupt Mask Register
A2	Read Bank C ICW2
A8	Read/Write IRQ16 Vector Register
A9	Read/Write IRQ17 Vector Register
AA	Read/Write IRQ18 Vector Register
AB	Read/Write IRQ19 Vector Register
AC	Read/Write IRQ20 Vector Register
AD	Read/Write IRQ21 Vector Register
AE	Read/Write IRQ22 Vector Register
AF	Read/Write IRQ23 Vector Register
30	Write Bank A ICW1, OCW2 or OCW3 Read Bank A Poll, Interrupt Request or In-Service Status Register
31	Write Bank A ICW2, ICW3, ICW4 or OCW1 Read Bank A Interrupt Mask Register
32	Read Bank A ICW2
38	Read/Write IRQ0 Vector Register
39	Read/Write IRQ1 Vector Register
3A	Read/Write IRQ1.5 Vector Register
3B	Read/Write IRQ3 Vector Register
3C	Read/Write IRQ4 Vector Register
3D	Reserved
3E	Reserved
3F	Read/Write IRQ7 Vector Register

APPENDIX B—Ports Listed by Function (Continued)

Port Address (HEX)	Description
PROGRAMMABLE INTERVAL TIMER	
40	Read/Write Counter 0 Register
41	Read/Write Counter 1 Register
42	Read/Write Counter 2 Register
43	Write Control Word Register I—Counter 0, 1, 2
44	Read/Write Counter 3 Register
47	Write Word Register II—Counter 3
CPU RESET	
64	Write CPU Reset Register (Data-1111XXX0H)
WAIT STATE GENERATOR	
72	Read/Write Wait State Register 0
73	Read/Write Wait State Register 1
74	Read/Write Wait State Register 2
75	Read/Write Refresh Wait State Register
DRAM REFRESH CONTROLLER	
1C	Read/Write Refresh Control Register
INTERNAL CONTROL AND DIAGNOSTIC PORTS	
61	Write Internal Control Port
80	Read/Write Internal Diagnostic Port 0
88	Read/Write Internal Diagnostic Port 1
RELOCATION REGISTER	
7F	Read/Write Relocation Register
INTEL RESERVED PORTS	
2A	Reserved
3D	Reserved
3E	Reserved
45	Reserved
46	Reserved
76	Reserved
77	Reserved
7D	Reserved
7E	Reserved
CC	Reserved
CD	Reserved

APPENDIX C

Pin Descriptions

The 82380 provides all of the signals necessary to interface it to an 80386 processor. It has separate 32-bit address and data buses. It also has a set of control signals to support operation as a bus master or a bus slave. Several special function signals exist on the 82380 for interfacing the system support peripherals to their respective system counterparts. Following are the definitions of the individual pins of the 82380. These brief descriptions are provided as a reference. Each signal is further defined within the sections which describe the associated 82380 function.

A2-A31 I/O ADDRESS BUS

This is the 32-bit address bus. The addresses are doubleword memory and I/O addresses. These are three-state signals which are active only during Master mode. The address lines should be connected directly to the 80386's local bus.

BE0# I/O BYTE-ENABLE 0

BE0# active indicates that data bits D0–D7 are being accessed or are valid. It is connected directly to the 80386's BE0#. The byte enable signals are active outputs when the 82380 is in the Master mode.

BE1# I/O BYTE-ENABLE 1

BE1# active indicates that data bits D8–D15 are being accessed or are valid. It is connected directly to the 80386's BE1#. The byte enable signals are active only when the 82380 is in the Master mode.

BE2# I/O BYTE-ENABLE 2

BE2# active indicates that data bits D15–D23 are being accessed or are valid. It is connected directly to the 80386's BE2#. The byte enable signals are active only when the 82380 is in the Master mode.

BE3# I/O BYTE-ENABLE 3

BE3# active indicates that data bits D24–D31 are being accessed or are valid. The byte enable signals are active only when the 82380 is in the Master mode. This pin should be connected directly to the 80386's BE3#. This pin is used for factory testing and must be low during reset. The 80386 drives BE3# low during reset.

D0–D31 I/O DATA BUS

This is the 32-bit data bus. These pins are active outputs during interrupt acknowledges, during Slave accesses, and when the 82380 is in the Master mode.

CLK2 I PROCESSOR CLOCK

This pin must be connected to CLK2. The 82380 monitors the phase of this clock in order to remain synchronized with the 80386. This clock drives all of the internal synchronous circuitry.

D/C# I/O DATA/CONTROL

D/C# is used to distinguish between 80386 control cycles and DMA or 80386 data access cycles. It is active as an output only in the Master mode.

W/R# I/O WRITE/READ

W/R# is used to distinguish between write and read cycles. It is active as an output only in the Master mode.

M/IO# I/O MEMORY/IO

M/IO# is used to distinguish between memory and IO accesses. It is active as an output only in the Master mode.

ADS# I/O ADDRESS STATUS

This signal indicates presence of a valid address on the address bus. It is active as output only in the Master mode. ADS# is active during the first T-state where addresses and control signals are valid.

NA# I NEXT ADDRESS

Asserted by a peripheral or memory to begin a pipelined address cycle. This pin is monitored only while the 82380 is in the Master mode. In the Slave mode, pipelining is determined by the current and past status of the ADS# and READY# signals.

HOLD O HOLD REQUEST

This is an active-high signal to the 80386 to request control of the system bus. When control is granted, the 80386 activates the hold acknowledge signal (HLDA).

HLDA I HOLD ACKNOWLEDGE

This input signal tells the DMA controller that the 80386 has relinquished control of the system bus to the DMA controller.

DREQ (0-3, 5-7) I DMA REQUEST

The DMA Request inputs monitor requests from peripherals requiring DMA service. Each of the eight DMA channels has one DREQ input. These active-high inputs are internally synchronized and prioritized. Upon reset, channel 0 has the highest priority and channel 7 the lowest.

DREQ4/IRQ9# I DMA/INTERRUPT REQUEST

This is the DMA request input for channel 4. It is also connected to the interrupt controller via interrupt request 9. This internal connection is available for DMA channel 4 only. The interrupt input is active low and can be programmed as either edge or level triggered. Either function can be masked by the appropriate mask register. Priorities of the DMA channel and the interrupt request are not related but follow the rules of the individual controllers.

Note that this pin has a weak internal pull-up. This causes the interrupt request to be inactive, but the DMA request will be active if there is no external connection made. Most applications will require that either one or the other of these functions be used, but not both. For this reason, it is advised that DMA channel 4 be used for transfers where a software request is more appropriate (such as memory-to-memory transfers). In such an application, DREQ4 can be masked by software, freeing IRQ9# for other purposes.

EOP# I/O END OF PROCESS

As an output, this signal indicates that the current Requester access is the last access of the currently operating DMA channel. It is activated when Terminal Count is reached. As an input, it signals the DMA channel to terminate the current buffer and proceed to the next buffer, if one is available. This signal may be programmed as an asynchronous or synchronous input.

EOP# must be connected to a pull-up resistor. This will prevent erroneous external requests for termination of a DMA process.

EDACK (0-2) O ENCODED DMA ACKNOWLEDGE

These signals contain the encoded acknowledgement of a request for DMA service by a peripheral. The binary code formed by the three signals indicates which channel is active. Channel 4 does not have a DMA acknowledge. The inactive state is indicated by the code 100. During a Requester access, EDACK presents the code for the active DMA channel. During a Target access, EDACK presents the inactive code 100.

IRQ (11-23)# I INTERRUPT REQUEST

These are active low interrupt request inputs. The inputs can be programmed to be edge or level sensitive. Interrupt priorities are programmable as either fixed or rotating. These inputs have weak internal pull-up resistors. Unused interrupt request inputs should be tied inactive externally.

INT O INTERRUPT OUT

INT signals the 80386 that an interrupt request is pending.

CLKIN I TIMER CLOCK INPUT

This is the clock input signal to all of the 82380's programmable timers. It is independent of the system clock input (CLK2).

TOUT1/REF# O TIMER 1 OUTPUT/REFRESH

This pin is software programmable as either the direct output of Timer 1, or as the indicator of a refresh cycle in progress. As REF#, this signal is active during the memory read cycle which occurs during refresh.

TOUT2#/IRQ3# I/O TIMER 2 OUTPUT/INTERRUPT REQUEST3

This is the inverted output of Timer 2. It is also connected directly to interrupt request 3. External hardware can use IRQ3# if Timer 2 is programmed as OUT=0 (TOUT2# = 1)

TOUT3# O TIMER 3 OUTPUT

This is the inverted output of Timer 3.

READY# I READY INPUT

This active-low input indicates to the 82380 that the current bus cycle is complete. READY is sampled by the 82380 both while it is in the Master mode, and while it is in the Slave mode.

WSC (0-1) I WAIT STATE CONTROL

WSC0 AND WSC1 are inputs used by the Wait-State Generator to determine the number of wait states required by the currently accessed memory or I/O. The binary code on these ins, combined with the M/IO# signal, selects an internal register in which a wait-state count is stored. The combination WSC = 11 disables the wait-state generator.

READYO# O READY OUTPUT

This is the synchronized output of the wait-state generator. It is also valid during 80386 accesses to the 82380 in the Slave Mode when the 82380 requires wait states. READYO# should feed directly the 80386's READY# input.

RESET I RESET

This synchronous input serves to initialize the state of the 82380 and provides basis for the CPURST output. RESET must be held active for at least 15 CLK2 cycles in order to guarantee the state of the 82380. After Reset, the 82380 is in the Slave mode with all outputs except timers and interrupts in their inactive states. The state of the timers and interrupt controller must be initialized through software. This input must be active for the entire time required by the 80386 to guarantee proper reset.

CPURST O CPU RESET

CPURST provides a synchronized reset signal for the CPU. It is activated in the event of a software reset command, an 80386 shut-down detect, or a hardware reset via the RESET pin. The 82380 holds CPURST active for 62 clocks in response to either a software reset command or a shut-down detection. Otherwise CPURST reflects the RESET input.

VCC +5V input power
VSS Ground

Table C-1. Wait-State Select Inputs

Port Address	Wait-State Registers				Select Inputs	
	D7	D4	D3	D0	WSC1	WSC0
72H	Memory 0		I/O 0		0	0
73H	Memory 1		I/O 1		0	1
74H	Memory 2		I/O 2		1	1
	DISABLED				1	1
M/IO#	1		0			

APPENDIX D

82380 System Notes

82380 TIMER UNIT SYSTEM NOTES

The 82380 DMA controller with Integrated System Peripherals is functionally inconsistent with the data sheet. This document explains the behavior of the 82380 Timer Unit and outlines subsequent limitations of the timer unit. This document also provides recommended workarounds.

Overview

There are two areas in which the 82380 timer unit exhibits non-specified behavior:

1. Mode 0 operation
2. Write Cycles to the 82380 Timer Unit

1.0 MODE 0 OPERATION

1.1 Description

For Mode 0 operation, the 82380 timer is specified as follows in the Intel 1989 Microprocessor and Peripheral Handbook Vol. I Page 4-240:

"1. Writing the first byte disables counting, OUT is set LOW immediately . . ."

Due to mode 0 errata, this should read as follows:

"1. Writing the first byte sets OUT LOW immediately. If the counter has not yet expired, writing the first byte also disables counting. However, if the counter has expired, writing the first count **does not** disable counting, although OUT still behaves correctly (set LOW immediately)."

1.2 Consequences

Software errors will occur if algorithms depend on the 82380 timer unit to stop counting after writing the first byte. Thus, software that is based on the 8254 core will not function reliably on the 82380 timer unit.

Note, however, that the external signal of the timer behaves correctly.

1.3 Solution

As long as software algorithms are aware of this behavior, there should be no problems, as the external signal behaves correctly.

1.4 Long Term Plans

Currently, Intel has no plans to fix this behavior of the 82380 timer unit.

2.0 WRITE CYCLES TO THE 82380 TIMER UNIT

This errata applies only to SLAVE WRITE cycles to the 82380 timer unit. During these cycles, the data being written into the 82380 timer unit may be corrupted if CLKIN is not inhibited during a certain "window" of the write cycle.

2.1 Description

Please refer to Figure 1.

During write cycles to the 82380 timer unit, the 82380 translates the 386DX interface signals such as ADS#, W/R#, M/IO#, and D/C# into several internal signals that control the operation of the internal sub-blocks (e.g., Timer Unit).

The 82380 timer unit is controlled by such internal signals. These internal signals are generated and sampled with respect to two separate clock signals: CLK2 (the system clock) and CLKIN (the 82380 timer unit clock).

Since the CLKIN and CLK2 clock signals are used internally to generate control signals for the interface to the timer unit, some timing parameters must be met in order for the interface logic to function properly.

Those timing parameters are met by inhibiting the CLKIN signal for a specific window during Write Cycles to the 82380 Timer Unit.

The CLKIN signal must be inhibited using external logic, as the GATE function of the 82380 timer unit is not guaranteed to totally inhibit CLKIN.

2.2 Consequences

This CLKIN inhibit circuitry guarantees proper write cycles to the 82380 timer unit.

Without this solution, write cycles to the 82380 timer unit could place corrupted data into the timer unit registers. This, in turn, could yield inaccurate results and improper timer operation.

The proposed solution would involve a hardware modification for existing systems.

2.3 Solution

A timing waveform (Figure 2) shows the specific window during which CLKIN must be inhibited. Please note that CLKIN must only be inhibited during the window shown in Figure 2. This window is defined by two AC timing parameters:

$$t_a = 9 \text{ ns}$$

$$t_b = 28 \text{ ns}$$

The proposed solution provides a certain amount of system "guardband" to make sure that this window is avoided.

PAL equations for a suggested workaround are also included. Please refer to the comments in the PAL codes for stated assumptions of this particular workaround. A state diagram (Figure 3) is provided to help clarify how this PAL is designed.

Figure 4 shows how this PAL would fit into a system workaround. In order to show the effect of this workaround on the CLKIN signal, Figure 5 shows how CLKIN is inhibited. Note that you must still meet the CLKIN AC timing parameters (e.g., t_{47} (min), t_{48} (min)) in order for the timer unit to function properly.

Please note that this workaround has not been tested. It is provided as a suggested solution. Actual solutions will vary from system to system.

2.4 Long Term Plans

Intel has no plans to fix this behavior in the 82380 timer unit.

```

module Timer_82380_Fix
flag '-r2','-q2','-fl', '-t4', '-w1,3,6,5,4,16,7,12,17,18,15,14'
title '82380 Timer Unit CLKIN
      INHIBIT signal PAL Solution '
Timer_Unit_Fix device 'P16R6';

"This PAL inhibits the CLKIN signal (that comes from an oscillator)
"during Slave Writes to the 82380 Timer unit.
"
"ASSUMPTION: This PAL assumes that an external system address
" decoder provides a signal to indicate that an 82380
" Timer Unit access is taking place. This input
" signal is called TMR in this PAL. This PAL also
" assumes that this TMR signal occurs during a
" specific T-State. Please see Figure 3 of this
" document to see when this signal is expected to
" be active by this PAL.
"
"NOTE: This PAL does not support pipelined 82380 SLAVE
" cycles.
"
"(c) Intel Corporation 1989. This PAL is provided as a proposed
"method of solving a certain 82380 Timer Unit problem. This PAL
"has not been tested or validated. Please validate this solution
"for your system and application.
"

```

"Input Pins"

CLK2	pin	1;	"System Clock
RESET	pin	2;	"Microprocessor RESET signal
TMR	pin	3;	"Input from Address Decoder, indicating "an access to the timer unit of the "82380.
!RDY	pin	4;	"End of Cycle indicator
!ADS	pin	5;	"Address and control strobe
CLK	pin	6;	"PHI2 clock
W_R	pin	7;	"Write/Read Signal"
nc1	pin	8;	"No Connect 0"
nc3	pin	9;	"No Connect 1"
GNDa	pin	10;	"Tied to ground, documentation only
GNDb	pin	11;	"Output enable, documentation only
CLKIN_IN	pin	12;	"Input-CLKIN directly from oscillator

"Output Pins"

Q_0	pin	18;	"Internal signal only, fed back to "PAL logic"
CLKIN_OUT	pin	17;	"CLKIN signal fed to 82380 Timer Unit
INHIBIT	pin	16;	"CLKIN Inhibit signal
S0	pin	15;	"Unused State Indicator Pin
S1	pin	14;	"Unused State Indicator Pin

"Declarations"

```
Valid_ADS = ADS & CLK      ; "ADS# sampled in PHI1 of 386DX T-State
Valid_RDY = RDY & CLK      ; "RDY# sampled in PHI1 of 386DX T-State
Timer_Acc = TMR & CLK      ; "Timer Unit Access, as provided by
                           "external Address Decoder "
```

State_Diagram [INHIBIT, S1, S0]

```
state 000:   if RESET then 000
             else if Valid_ADS & W_R then 001
             else 000;

state 001:   if RESET then 000
             else if Timer_Acc then 010
             else if !Timer_Acc then 000
             else 001;

state 010:   if RESET then 000
             else if CLK then 110
             else 010;

state 110:   if RESET then 000
             else if CLK then 111
             else 110;
```

```

state 111:    if RESET then 000
              else if CLK then 011
              else 111;

state 011:    if RESET then 000
              else if Valid_RDY then 000
              else 011;

state 100:    if RESET then 000
              else 000;

state 101:    if RESET then 000
              else 000;

```

EQUATIONS

```

Q_0 := CLKIN_IN ; "Latched incoming clock. This signal is used
                  "internally to feed into the MUX-ing logic"

CLKIN_OUT := (INHIBIT & CLKIN_OUT & !RESET)
             + (!INHIBIT & Q_0 & !RESET);

                  "Equation for CLKIN_OUT. This
                  "feeds directly to the 82380 Timer Unit."

```

END

Page 1

ABEL(tm) 3.10 - Document Generator 30-June 89 03:17
PM

82380 Timer Unit CLKIN
INHIBIT signal PAL Solution
Equations for Module Timer_82380_Fix

Device Timer_Unit_Fix

- Reduced Equations:

```

!INHIBIT := (!CLK & !INHIBIT # CLK & SO # RESET # !S1);

```

```

!S1 := (RESET
       # INHIBIT & !S1
       # CLK & !INHIBIT & !~RDY & SO & S1
       # !CLK & !S1
       # !S1 & !TMR
       # !SO & !S1);

```

```

!SO := (RESET
       # INHIBIT & !S1
       # CLK & !INHIBIT & !~RDY & S1
       # !CLK & !SO
       # !INHIBIT & !SO & S1
       # SO & !S1
       # !S1 & !W_R
       # ~ADS & !S1);

```

```
!Q_0 := (!CLKIN_IN);
```

```
!CLKIN_OUT := (RESET # !CLKIN_OUT & INHIBIT # !INHIBIT & !Q_0);
```

Page 2

ABEL(tm) 3.10 - Document Generator 30-June 89 03:17

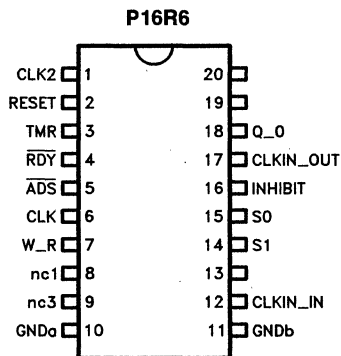
PM

82380 Timer Unit CLKIN

INHIBIT signal PAL Solution

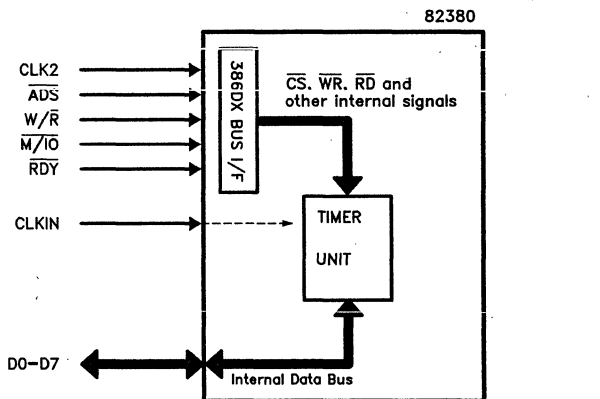
Chip diagram for Module Timer_82380_Fix

Device Timer_Unit_Fix



290128-B7

end of module Timer_82380_Fix



290128-B8

Figure 1. Translation of 386DX Signals to Internal 82380 Timer Unit Signals

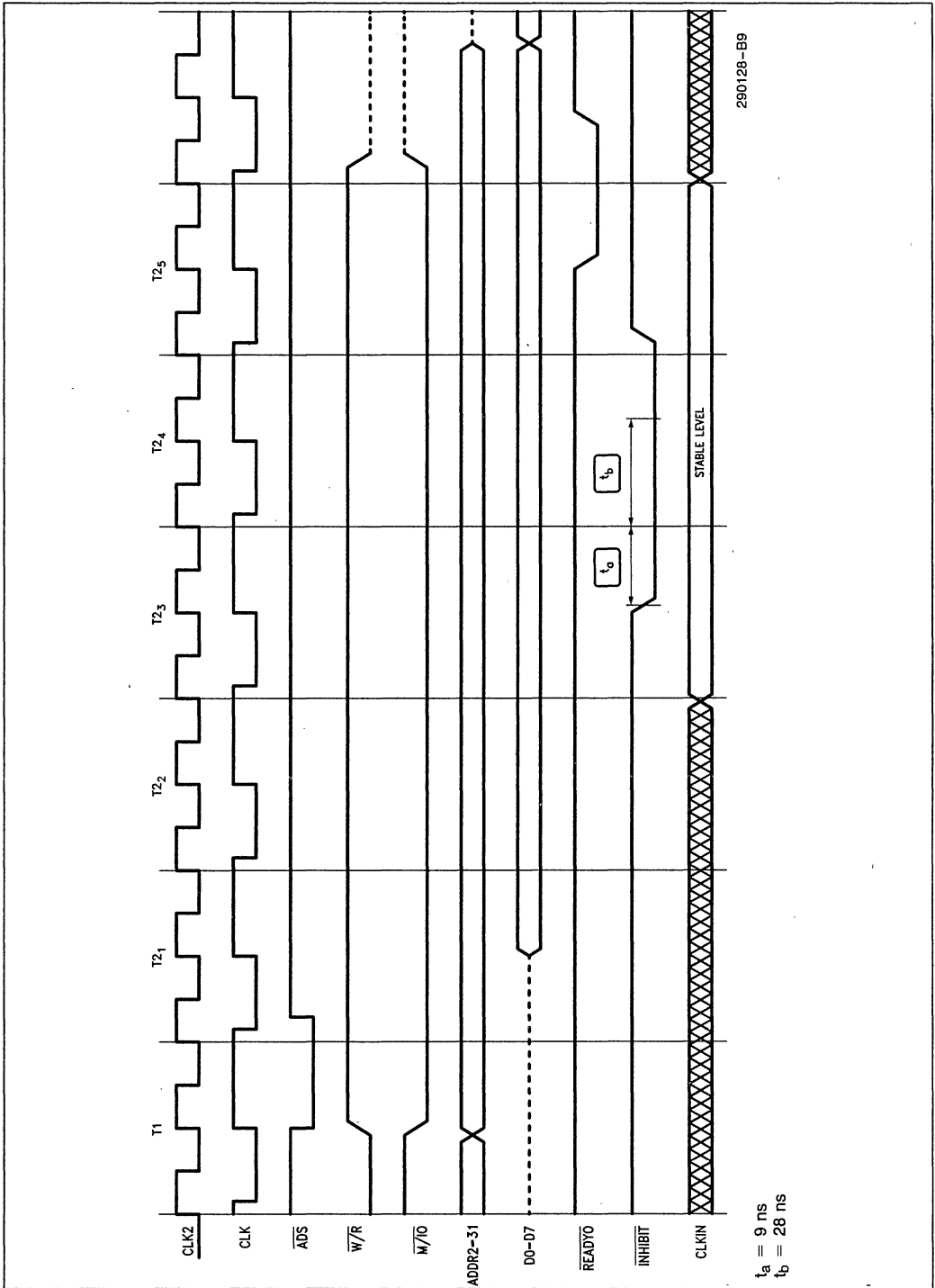


Figure 2. 82380 Timer Unit Write Cycle

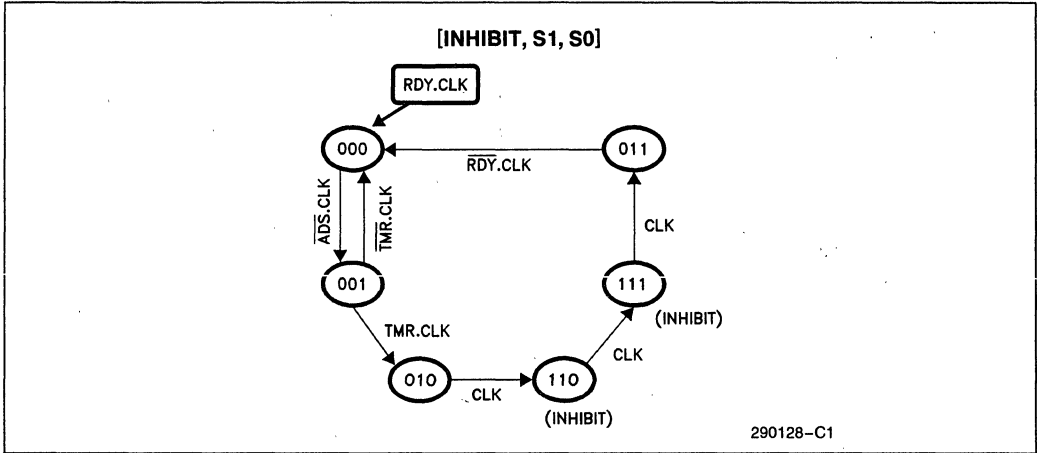


Figure 3. State Diagram for Inhibit Signal

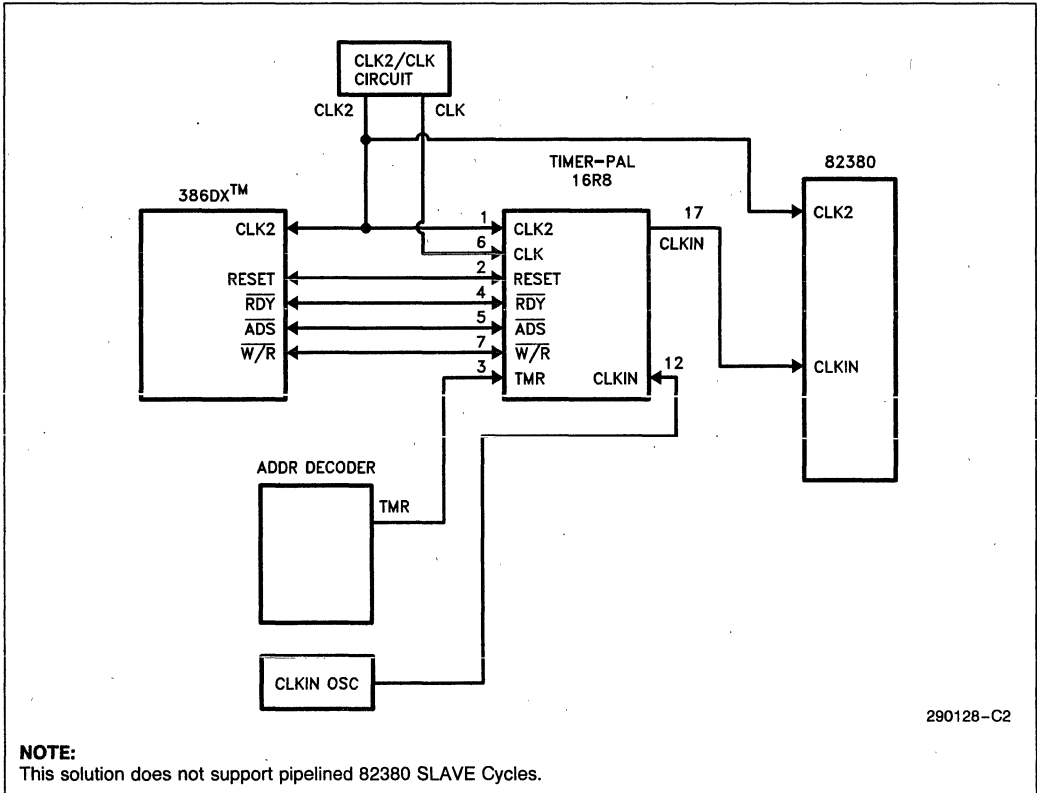


Figure 4. System with 82380 Timer Unit "Inhibit" Circuitry

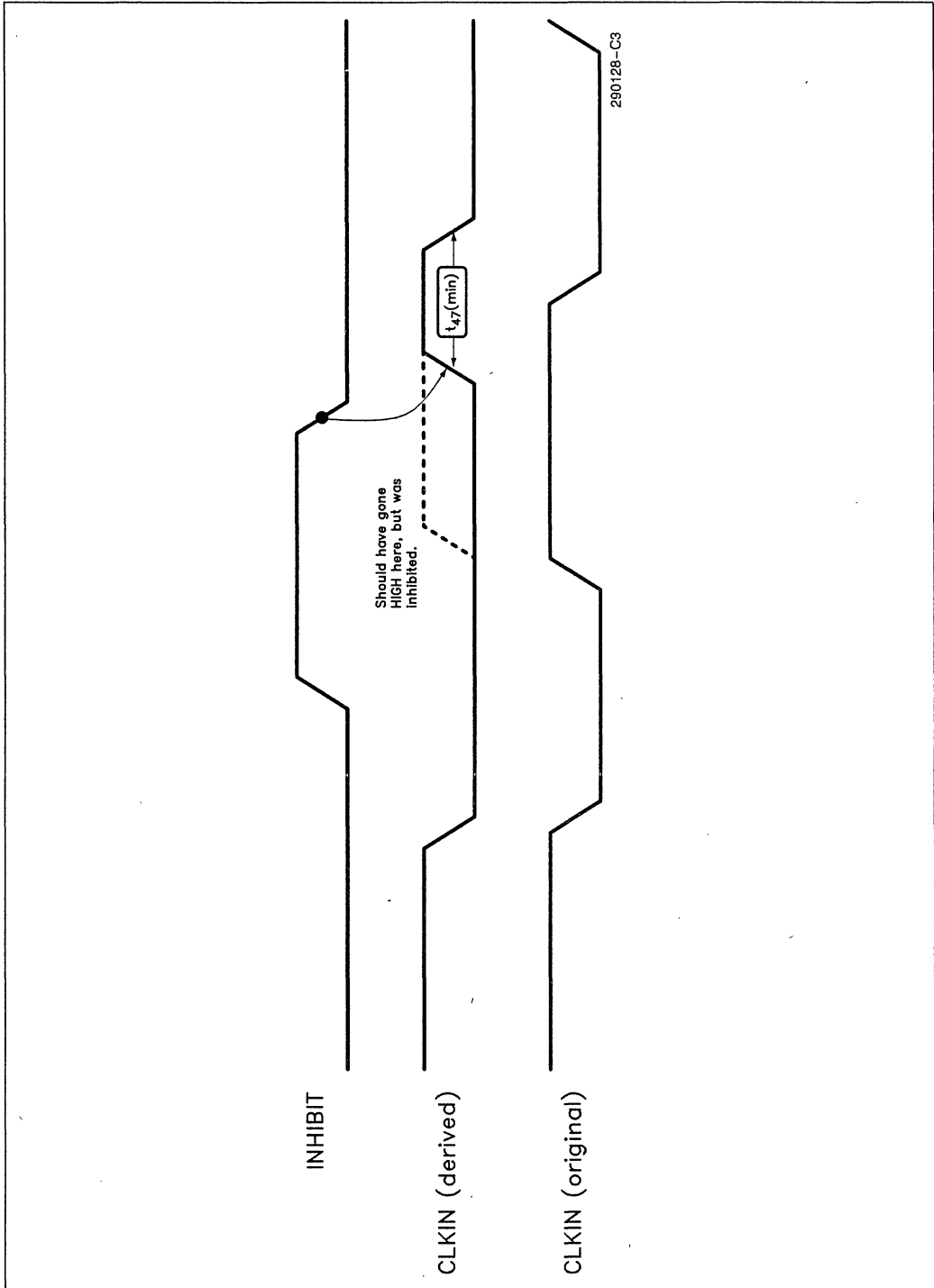
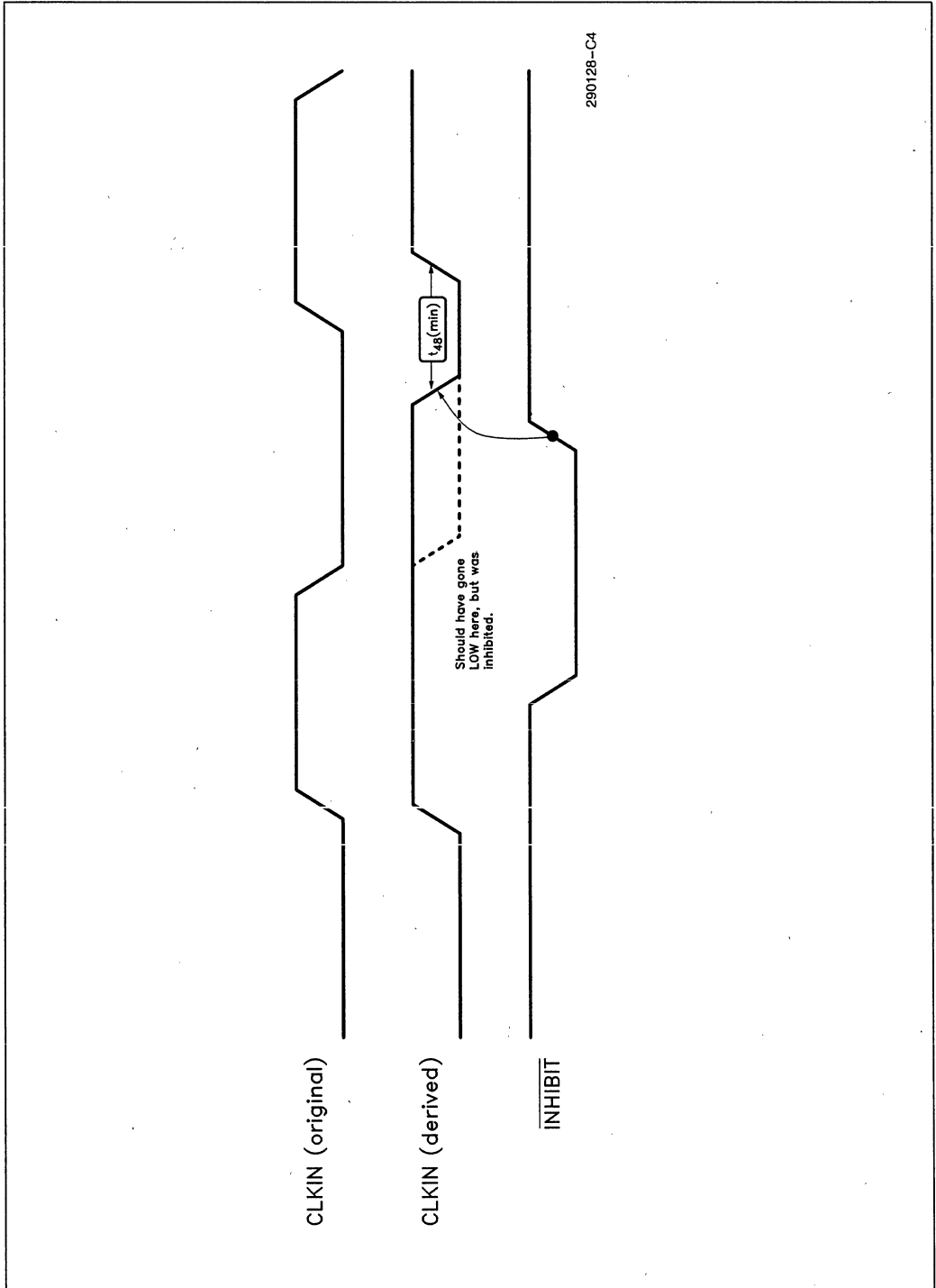


Figure 5(a). Inhibited CLKIN in an 82380 Timer Unit and CLKIN Minimum HIGH Time



290128-C4

Figure 5(b). Inhibited CLKIN in an 82380 Timer Unit and CLKIN Minimum LOW Time

82380 DATA SHEET REVISION HISTORY

Changes in this revision:

Figure 4-1: Added details about IRQ3# and IRQ2#/IRQ1.5#.

Section 5.2.1: Added note referring reader to Appendix D (System Notes).

Table 13-2: Changed V_{IHC} MIN to $V_{CC} - 0.8V$.

Figure 13-1: Changed signal names to reflect accurate drive levels and measurement points for those signals.

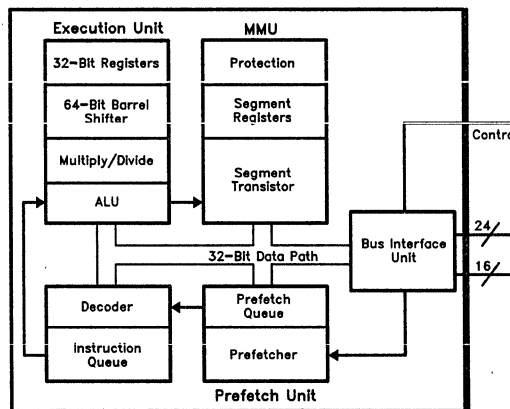
Appendix D: Added this appendix to explain the restrictions on the CLKIN signal of the 82380 Timer Unit.

376™ HIGH PERFORMANCE 32-BIT EMBEDDED PROCESSOR

- **Full 32-Bit Internal Architecture**
 - 8-, 16-, 32-Bit Data Types
 - 8 General Purpose 32-Bit Registers
 - Extensive 32-Bit Instruction Set
- **High Performance 16-Bit Data Bus**
 - 16 MHz CPU Clock
 - Two-Clock Bus Cycles
 - 16 Mbytes/Sec Bus Bandwidth
- **16 Mbyte Physical Memory Size**
- **High Speed Numerics Support with the 80387SX**
- **Low System Cost with the 82370 Integrated System Peripheral**
- **On-Chip Debugging Support Including Break Point Registers**
- **Complete Intel Development Support**
 - C, PL/M, Assembler
 - ICETM-376, In-Circuit Emulator
 - iRMKTM Real Time Kernel
 - iSDMTM Debug Monitor
 - DOS Based Debug
- **Extensive Third-Party Support:**
 - Languages: C, Pascal, FORTRAN, BASIC and ADA*
 - Hosts: VMS*, UNIX*, MS-DOS*, and Others
 - Real-Time Kernels
- **High Speed CHMOS Technology**
- **Available in 100 Pin Plastic Quad Flat-Pack Package and 88-Pin Pin Grid Array**
(See Packaging Outlines and Dimensions #231369)

INTRODUCTION

The 376 32-bit embedded processor is designed for high performance embedded systems. It provides the performance benefits of a highly pipelined 32-bit internal architecture with the low system cost associated with 16-bit hardware systems. The 80376 is based on the 80386 and offers a high degree of compatibility with the 80386. All 80386 32-bit programs not dependent on paging can be executed on the 80376 and all 80376 programs can be executed on the 80386. All 32-bit 80386 language translators can be used for software development. With proper support software, any 80386-based computer can be used to develop and test 80376 programs. In addition, any 80386-based PC-AT* compatible computer can be used for hardware prototyping for designs based on the 80376 and its companion product the 82370.



240182-48

80376 Microarchitecture

Intel, iRMK, ICE, 376, 386, Intel386, iSDM, Intel1376 are trademarks of Intel Corp.

*UNIX is a registered trademark of AT&T.

ADA is a registered trademark of the U.S. Government, Ada Joint Program Office.

PC-AT is a registered trademark of IBM Corporation.

VMS is a trademark of Digital Equipment Corporation.

MS-DOS is a trademark of MicroSoft Corporation.

1.0 PIN DESCRIPTION

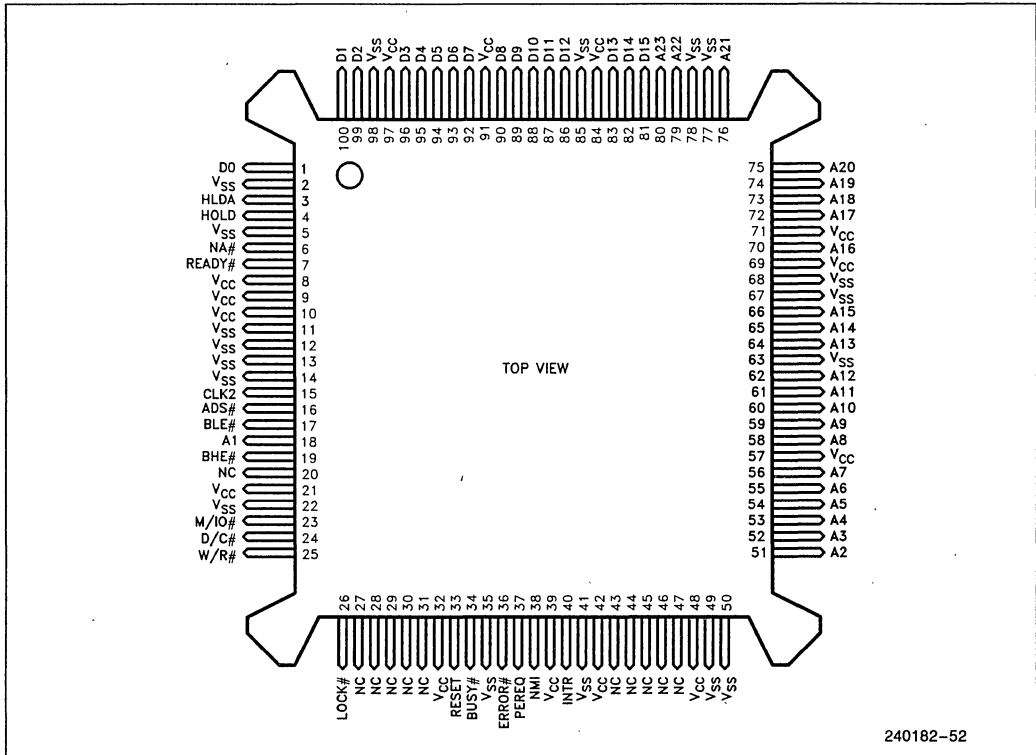


Figure 1.1. 80376 100-Pin Quad Flat-Pack Pin Out (Top View)

Table 1.1. 100-Pin Plastic Quad Flat-Pack Pin Assignments

Address	Data	Control	N/C	Vcc	Vss		
A ₁	D ₀	1	ADS #	16	20	8	2
A ₂	D ₁	100	BHE #	19	27	9	5
A ₃	D ₂	99	BLE #	17	28	10	11
A ₄	D ₃	96	BUSY #	34	29	21	12
A ₅	D ₄	95	CLK2	15	30	32	13
A ₆	D ₅	94	D/C #	24	31	39	14
A ₇	D ₆	93	ERROR #	36	43	42	22
A ₈	D ₇	92	HLDA	3	44	48	35
A ₉	D ₈	90	HOLD	4	45	57	41
A ₁₀	D ₉	89	INTR	40	46	69	49
A ₁₁	D ₁₀	88	LOCK #	26	47	71	50
A ₁₂	D ₁₁	87	M/IO #	23		84	63
A ₁₃	D ₁₂	86	NA #	6		91	67
A ₁₄	D ₁₃	83	NMI	38		97	68
A ₁₅	D ₁₄	82	PEREQ	37			77
A ₁₆	D ₁₅	81	READY #	7			78
A ₁₇			RESET	33			85
A ₁₈			W/R #	25			98
A ₁₉							
A ₂₀							
A ₂₁							
A ₂₂							
A ₂₃							

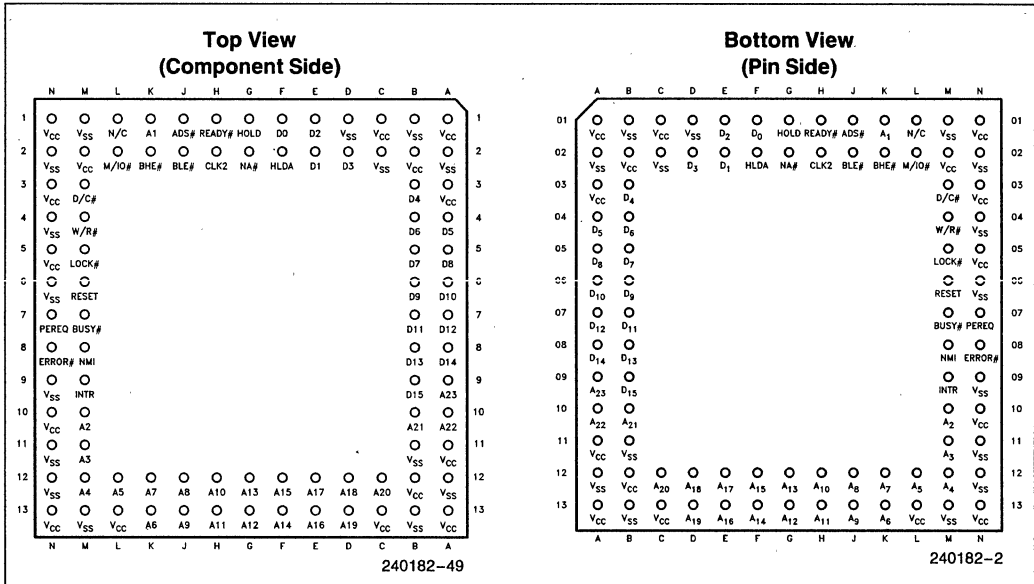


Figure 1.2. 80376 88-Pin Grid Array Pin Out

Table 1.2. 88-Pin Grid Array Pin Assignments

Pin	Label	Pin	Label	Pin	Label	Pin	Label
2H	CLK2	12D	A18	2L	M/IO#	11A	VCC
9B	D15	12E	A17	5M	LOCK#	13A	VCC
8A	D14	13E	A16	1J	ADS#	13C	VCC
8B	D13	12F	A15	1H	READY#	13L	VCC
7A	D12	13F	A14	2G	NA#	1N	VCC
7B	D11	12G	A13	1G	HOLD	13N	VCC
6A	D10	13G	A12	2F	HLDA	11B	VSS
6B	D9	13H	A11	7N	PEREQ	2C	VSS
5A	D8	12H	A10	7M	BUSY#	1D	VSS
5B	D7	13J	A9	8N	ERROR#	1M	VSS
4B	D6	12J	A8	9M	INTR	4N	VSS
4A	D5	12K	A7	8M	NMI	9N	VSS
3B	D4	13K	A6	6M	RESET	11N	VSS
2D	D3	12L	A5	2B	VCC	2A	VSS
1E	D2	12M	A4	12B	VCC	12A	VSS
2E	D1	11M	A3	1C	VCC	1B	VSS
1F	D0	10M	A2	2M	VCC	13B	VSS
9A	A23	1K	A1	3N	VCC	13M	VSS
10A	A22	2J	BLE#	5N	VCC	2N	VSS
10B	A21	2K	BHE#	10N	VCC	6N	VSS
12C	A20	4M	W/R#	1A	VCC	12N	VSS
13D	A19	3M	D/C#	3A	VCC	1L	N/C

The following table lists a brief description of each pin on the 80376. The following definitions are used in these descriptions:

- # The named signal is active LOW.
- I Input signal.
- O Output signal.
- I/O Input and Output signal.
- No electrical connection.

Symbol	Type	Name and Function
CLK2	I	CLK2 provides the fundamental timing for the 80376. For additional information see Clock in Section 4.1.
RESET	I	RESET suspends any operation in progress and places the 80376 in a known reset state. See Interrupt Signals in Section 4.1 for additional information.
D ₁₅ -D ₀	I/O	DATA BUS inputs data during memory, I/O and interrupt acknowledge read cycles and outputs data during memory and I/O write cycles. See Data Bus in Section 4.1 for additional information.
A ₂₃ -A ₁	O	ADDRESS BUS outputs physical memory or port I/O addresses. See Address Bus in Section 4.1 for additional information.
W/R#	O	WRITE/READ is a bus cycle definition pin that distinguishes write cycles from read cycles. See Bus Cycle Definition Signals in Section 4.1 for additional information.
D/C#	O	DATA/CONTROL is a bus cycle definition pin that distinguishes data cycles, either memory or I/O, from control cycles which are: interrupt acknowledge, halt, and instruction fetching. See Bus Cycle Definition Signals in Section 4.1 for additional information.
M/IO#	O	MEMORY I/O is a bus cycle definition pin that distinguishes memory cycles from input/output cycles. See Bus Cycle Definition Signals in Section 4.1 for additional information.
LOCK#	O	BUS LOCK is a bus cycle definition pin that indicates that other system bus masters are denied access to the system bus while it is active. See Bus Cycle Definition Signals in Section 4.1 for additional information.
ADS#	O	ADDRESS STATUS indicates that a valid bus cycle definition and address (W/R#, D/C#, M/IO#, BHE#, BLE# and A ₂₃ -A ₁) are being driven at the 80376 pins. See Bus Control Signals in Section 4.1 for additional information.
NA#	I	NEXT ADDRESS is used to request address pipelining. See Bus Control Signals in Section 4.1 for additional information.
READY#	I	BUS READY terminates the bus cycle. See Bus Control Signals in Section 4.1 for additional information.
BHE#, BLE#	O	BYTE ENABLES indicate which data bytes of the data bus take part in a bus cycle. See Address Bus in Section 4.1 for additional information.
HOLD	I	BUS HOLD REQUEST input allows another bus master to request control of the local bus. See Bus Arbitration Signals in Section 4.1 for additional information.

Symbol	Type	Name and Function
HLDA	O	BUS HOLD ACKNOWLEDGE output indicates that the 80376 has surrendered control of its local bus to another bus master. See Bus Arbitration Signals in Section 4.1 for additional information.
INTR	I	INTERRUPT REQUEST is a maskable input that signals the 80376 to suspend execution of the current program and execute an interrupt acknowledge function. See Interrupt Signals in Section 4.1 for additional information.
NMI	I	NON-MASKABLE INTERRUPT REQUEST is a non-maskable input that signals the 80376 to suspend execution of the current program and execute an interrupt acknowledge function. See Interrupt Signals in Section 4.1 for additional information.
BUSY#	I	BUSY signals a busy condition from a processor extension. See Coprocessor Interface Signals in Section 4.1 for additional information.
ERROR#	I	ERROR signals an error condition from a processor extension. See Coprocessor Interface Signals in Section 4.1 for additional information.
PEREQ	I	PROCESSOR EXTENSION REQUEST indicates that the processor extension has data to be transferred by the 80376. See Coprocessor Interface Signals in Section 4.1 for additional information.
N/C	—	NO CONNECT should always remain unconnected. Connection of a N/C pin may cause the processor to malfunction or be incompatible with future steppings of the 80376.
V _{CC}	I	SYSTEM POWER provides the +5V nominal D.C. supply input.
V _{SS}	I	SYSTEM GROUND provides 0V connection from which all inputs and outputs are measured.

2.0 ARCHITECTURE OVERVIEW

The 80376 supports the protection mechanisms needed by sophisticated multitasking embedded systems and real-time operating systems. The use of these protection mechanisms is completely optional. For embedded applications not needing protection, the 80376 can easily be configured to provide a 16 Mbyte physical address space.

Instruction pipelining, high bus bandwidth, and a very high performance ALU ensure short average instruction execution times and high system throughput. The 80376 is capable of execution at sustained rates of 2.5–3.0 million instructions per second.

The 80376 offers on-chip testability and debugging features. Four break point registers allow conditional or unconditional break point traps on code execution or data accesses for powerful debugging of even ROM based systems. Other testability features include self-test and tri-stating of output buffers during RESET.

The Intel 80376 embedded processor consists of a central processing unit, a memory management unit and a bus interface. The central processing unit con-

sists of the execution unit and instruction unit. The execution unit contains the eight 32-bit general registers which are used for both address calculation and data operations and a 64-bit barrel shifter used to speed shift, rotate, multiply, and divide operations. The instruction unit decodes the instruction opcodes and stores them in the decoded instruction queue for immediate use by the execution unit.

The Memory Management Unit (MMU) consists of a segmentation and protection unit. Segmentation allows the managing of the logical address space by providing an extra addressing component, one that allows easy code and data relocatability, and efficient sharing.

The protection unit provides four levels of protection for isolating and protecting applications and the operating system from each other. The hardware enforced protection allows the design of systems with a high degree of integrity and simplifies debugging.

Finally, to facilitate high performance system hardware designs, the 80376 bus interface offers address pipelining and direct Byte Enable signals for each byte of the data bus.

2.1 Register Set

The 80376 has twenty-nine registers as shown in Figure 2.1. These registers are grouped into the following six categories:

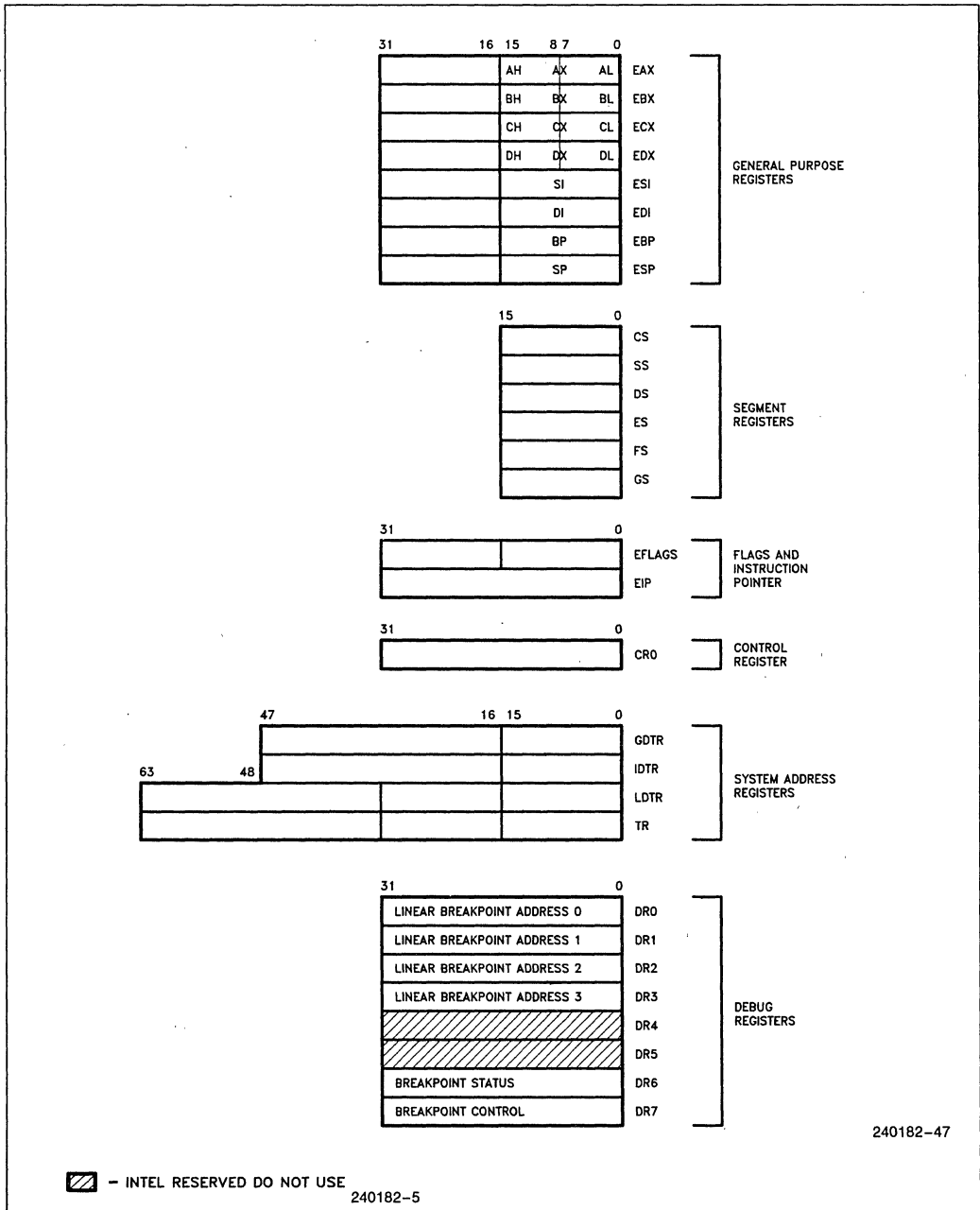


Figure 2.1. 80376 Base Architecture Registers

General Registers: The eight 32-bit general purpose registers are used to contain arithmetic and logical operands. Four of these (EAX, EBX, ECX and EDX) can be used either in their entirety as 32-bit registers, as 16-bit registers, or split into pairs of separate 8-bit registers.

Segment Registers: Six 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data.

Flags and Instruction Pointer Registers: These two 32-bit special purpose registers in Figure 2.1 record or control certain aspects of the 80376 processor state. The EFLAGS register includes status and control bits that are used to reflect the outcome of many instructions and modify the semantics of some instructions. The Instruction Pointer, called EIP, is 32 bits wide. The Instruction Pointer controls instruction fetching and the processor automatically increments it after executing an instruction.

Control Register: The 32-bit control register, CR0, is used to control Coprocessor Emulation.

System Address Registers: These four special registers reference the tables or segments supported by the 80376/80386 protection model. These tables or segments are:

- GDTR (Global Descriptor Table Register),
- IDTR (Interrupt Descriptor Table Register),
- LDTR (Local Descriptor Table Register),
- TR (Task State Segment Register).

Debug Registers: The six programmer accessible debug registers provide on-chip support for debugging. The use of the debug registers is described in Section 2.11 **Debugging Support**.

EFLAGS REGISTER

The flag Register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS, shown in Figure 2.2, control certain operations and indicate the status of the 80376 processor. The function of the flag bits is given in Table 2.1.

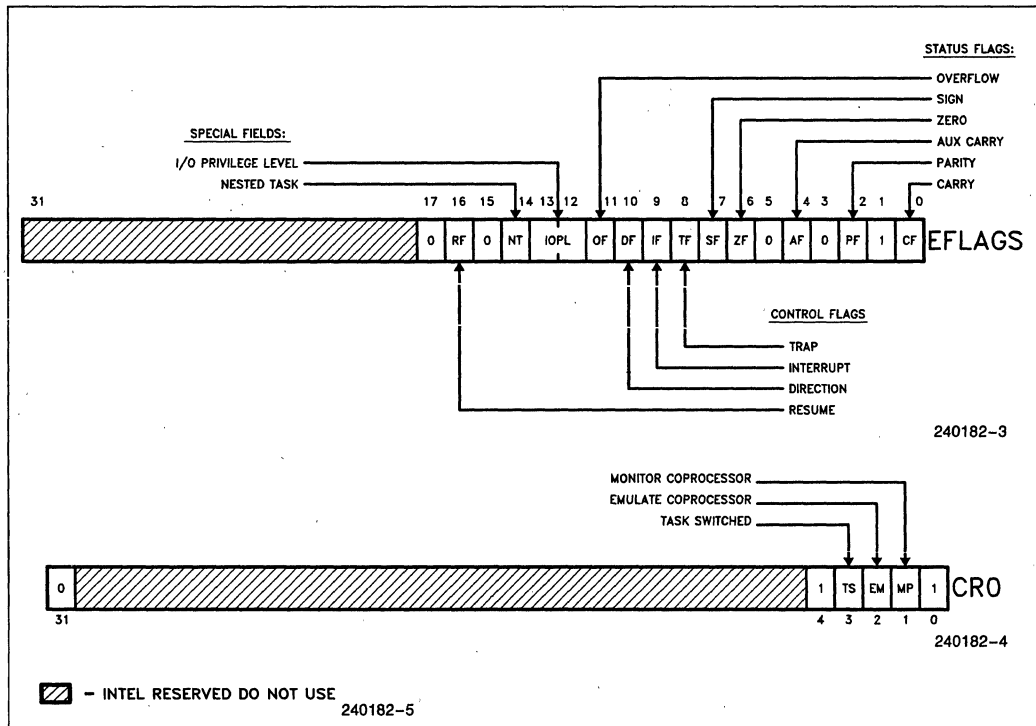


Figure 2.2. Status and Control Register Bit Functions

Table 2.1. Flag Definitions

Bit Position	Name	Function
0	CF	Carry Flag —Set on high-order bit carry or borrow; cleared otherwise.
2	PF	Parity Flag —Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise.
4	AF	Auxiliary Carry Flag —Set on carry from or borrow to the low order four bits of AL; cleared otherwise.
6	ZF	Zero Flag —Set if result is zero; cleared otherwise.
7	SF	Sign Flag —Set equal to high-order bit of result (0 if positive, 1 if negative).
8	TF	Single Step Flag —Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-Enable Flag —When set, external interrupts signaled on the INTR pin will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag —Causes string instructions to auto-increment (default) the appropriate index registers when cleared. Setting DF causes auto-decrement.
11	OF	Overflow Flag —Set if the operation resulted in a carry/borrow into the sign bit (high-order bit) of the result but did not result in a carry/borrow out of the high-order bit or vice-versa.
12, 13	IOPL	I/O Privilege Level —Indicates the maximum CPL permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O permission bit map. It also indicates the maximum CPL value allowing alteration of the IF bit.
14	NT	Nested Task —Indicates that the execution of the current task is nested within another task (see Task Switching).
16	RF	Resume Flag —Used in conjunction with debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. If set, any debug fault is ignored on the next instruction. It is reset at the successful completion of any instruction except IRET, POPF, and those instructions causing task switches.

CONTROL REGISTER

The 80376 has a 32-bit control register called CRO that is used to control coprocessor emulation. This register is shown in Figures, 2.1 and 2.2. The defined CRO bits are described in Table 2.2. Bits 0, 4 and 31 of CRO have fixed values in the 80376. These values cannot be changed. Programs that load CRO should always load bits 0, 4 and 31 with values previously there to be compatible with the 80386.

Table 2.2. CRO Definitions

Bit Position	Name	Function
1	MP	Monitor Coprocessor Extension —Allows WAIT instructions to cause a processor extension not present exception (number 7).
2	EM	Emulate Processor Extension —When set, this bit causes a processor extension not present exception (number 7) on ESC instructions to allow processor extension emulation.
3	TS	Task Switched —When set, this bit indicates the next instruction using a processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task (see Task Switching).

2.2 Instruction Set

The instruction set is divided into nine categories of operations:

- Data Transfer
- Arithmetic
- Shift/Rotate
- String Manipulation
- Bit Manipulation
- Control Transfer
- High Level Language Support
- Operating System Support
- Processor Control

These 80376 processor instructions are listed in Table 8.1 **80376 Instruction Set and Clock Count Summary**.

All 80376 processor instructions operate on either 0, 1, 2 or 3 operands; an operand resides in a register, in the instruction itself, or in memory. Most zero operand instructions (e.g. CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the 80376 has a 16-byte prefetch instruction queue an average of 5 instructions can be prefetched. The use of two operands permits the following types of common instructions:

- Register to Register
- Memory to Register
- Immediate to Register
- Memory to Memory
- Register to Memory
- Immediate to Memory

The operands are either 8-, 16- or 32-bit long.

2.3 Memory Organization

Memory on the 80376 is divided into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address. The address of a word or Dword is the byte address of the low-order byte. For maximum performance word and dword values should be at even physical addresses.

In addition to these basic data types the 80376 processor supports segments. Memory can be divided up into one or more variable length segments, which can be shared between programs.

ADDRESS SPACES

The 80376 has three types of address spaces: **logical**, **linear**, and **physical**. A **logical** address (also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, and DISPLACEMENT), discussed in Section 2.4 **Addressing Modes**, into an effective address.

Every selector has a **logical base** address associated with it that can be up to 32 bits in length. This 32-bit **logical base** address is added to either a 32-bit offset address or a 16-bit offset address (by using the **address length prefix**) to form a final 32-bit **linear** address. This final **linear** address is then truncated so that only the lower 24 bits of this address are used to address the 16 Mbytes physical memory address space. The **logical base** address is stored in one of two operating system tables (i.e. the Local Descriptor Table or Global Descriptor Table).

Figure 2.3 shows the relationship between the various address spaces.

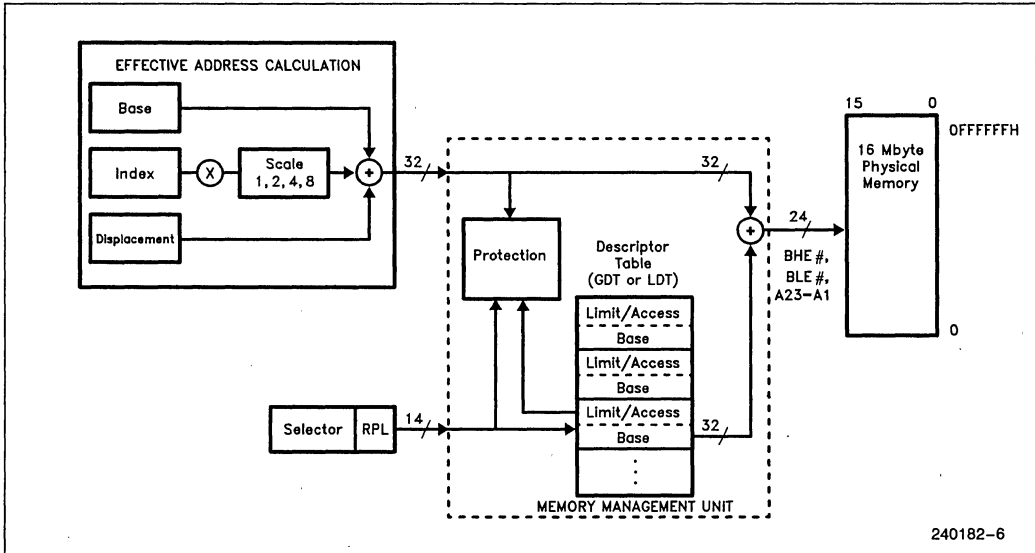


Figure 2.3. Address Translation

SEGMENT REGISTER USAGE

The main data structure used to organize memory is the segment. On the 80376, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments, code and data. The simplest use of segments is to have one code and data segment. Each segment is 16 Mbytes in size overlapping each other. This allows code and data to be directly addressed by the same offset.

In order to provide compact instruction encoding and increase processor performance, instructions do not need to explicitly specify which segment reg-

ister is used. The segment register is automatically chosen according to the rules of Table 2.3 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register, stack references use the SS register and instruction fetches use the CS register. The contents of the Instruction Pointer provide the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2.3. The override prefixes also allow the use of the ES, FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero. Further details of segmentation are discussed in Section 3.0 Architecture.

Table 2.3. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA Instructions	SS	None
Source of POP, POPA, POPF, IRET, RET Instructions	SS	None
Destination of STOS, MOVS, REP STOS, REP MOVS Instructions (DI is Base Register)	ES	None
Other Data References, with Effective Address Using Base Register of:		
[EAX]	DS	CS, SS, ES, FS, GS
[EBX]	DS	CS, SS, ES, FS, GS
[ECX]	DS	CS, SS, ES, FS, GS
[EDX]	DS	CS, SS, ES, FS, GS
[ESI]	DS	CS, SS, ES, FS, GS
[EDI]	DS	CS, SS, ES, FS, GS
[EBP]	SS	CS, SS, ES, FS, GS
[ESP]	SS	CS, SS, ES, FS, GS

2.4 Addressing Modes

The 80376 provides a total of 8 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

Two of the addressing modes provide for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8-, 16- or 32-bit general registers.

Immediate Operand Mode: The operand is included in the instruction as part of the opcode.

The remaining 6 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the seg-

ment base address and an effective address. The effective address is calculated by summing any combination of the following three address elements (see Figure 2.3):

DISPLACEMENT: an 8-, 16- or 32-bit immediate value following the instruction.

BASE: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area. Note that if the **Address Length Prefix** is used, only BX and BP can be used as a BASE register.

INDEX: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters. The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. The scaled index is especially useful for accessing arrays or structures. Note that if the **Address Length Prefix** is used, no Scaling is available and only the registers SI and DI can be used to INDEX.

Combinations of these 3 components make up the 6 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of BASE and INDEX components which requires one additional clock.

As shown in Figure 2.4, the effective address (EA) of an operand is calculated according to the following formula:

$$EA = \text{BASE}_{\text{Register}} + (\text{INDEX}_{\text{Register}} \times \text{scaling}) + \text{DISPLACEMENT}$$

1. Direct Mode: The operand's offset is contained as part of the instruction as an 8-, 16- or 32-bit DISPLACEMENT.

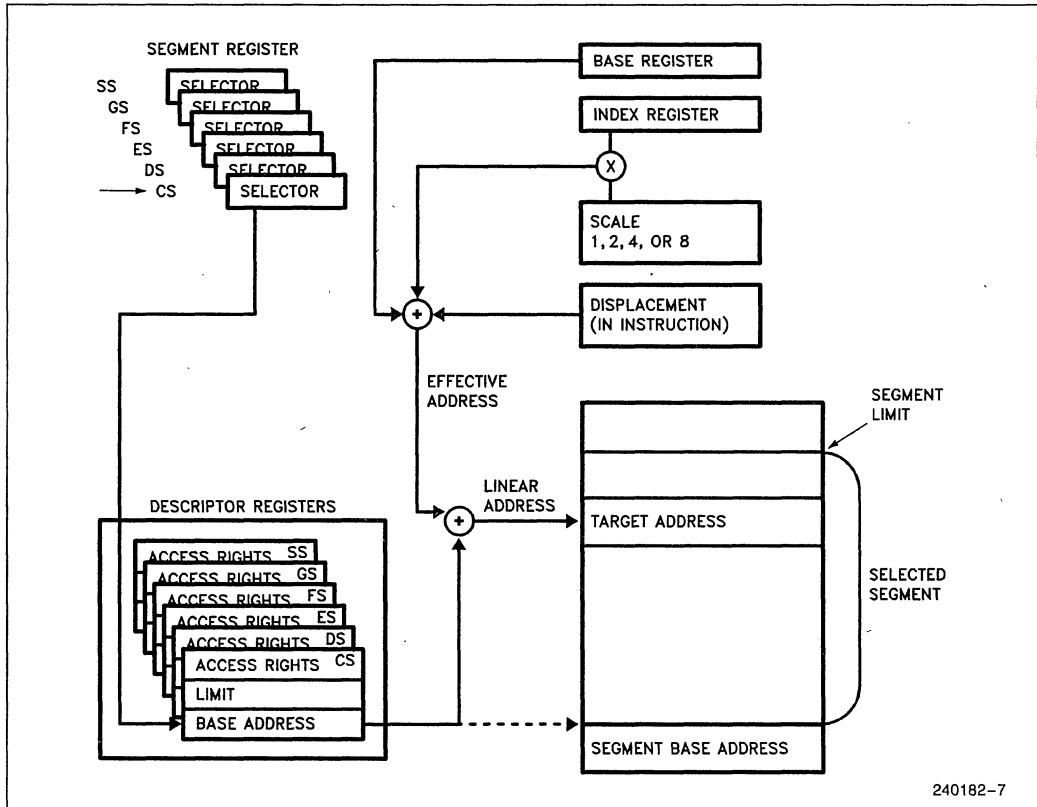
2. Register Indirect Mode: A BASE register contains the address of the operand.

3. Based Mode: A BASE register's contents is added to a DISPLACEMENT to form the operand's offset.

4. Scaled Index Mode: An INDEX register's contents is multiplied by a SCALING factor which is added to a DISPLACEMENT to form the operand's offset.

5. Based Scaled Index Mode: The contents of an INDEX register is multiplied by a SCALING factor and the result is added to the contents of a BASE register to obtain the operand's offset.

6. Based Scaled Index Mode with Displacement: The contents of an INDEX register are multiplied by a SCALING factor, and the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.



240182-7

Figure 2.4. Addressing Mode Calculations

GENERATING 16-BIT ADDRESSES

The 80376 executes code with a default length for operands and addresses of 32 bits. The 80376 is also able to execute operands and addresses of 16 bits. This is specified through the use of override prefixes. Two prefixes, the **Operand Length Prefix** and the **Address Length Prefix**, override the default 32-bit length on an individual instruction basis. These prefixes are automatically added by assem-

blers. The Operand Length and Address Length Prefixes can be applied separately or in combination to any instruction.

The 80376 normally executes 32-bit code and uses either 8- or 32-bit displacements, and any register can be used as based or index registers. When executing 16-bit code (by prefix overrides), the displacements are either 8 or 16 bits, and the base and index register conform to the 16-bit model. Table 2.4 illustrates the differences.

Table 2.4. BASE and INDEX Registers for 16- and 32-Bit Addresses

	16-Bit Addressing	32-Bit Addressing
BASE REGISTER	BX, BP	Any 32-Bit GP Register
INDEX REGISTER	SI, DI	Any 32-Bit GP Register except ESP
SCALE FACTOR	None	1, 2, 4, 8
DISPLACEMENT	0, 8, 16 Bits	0, 8, 32 Bits

2.5 Data Types

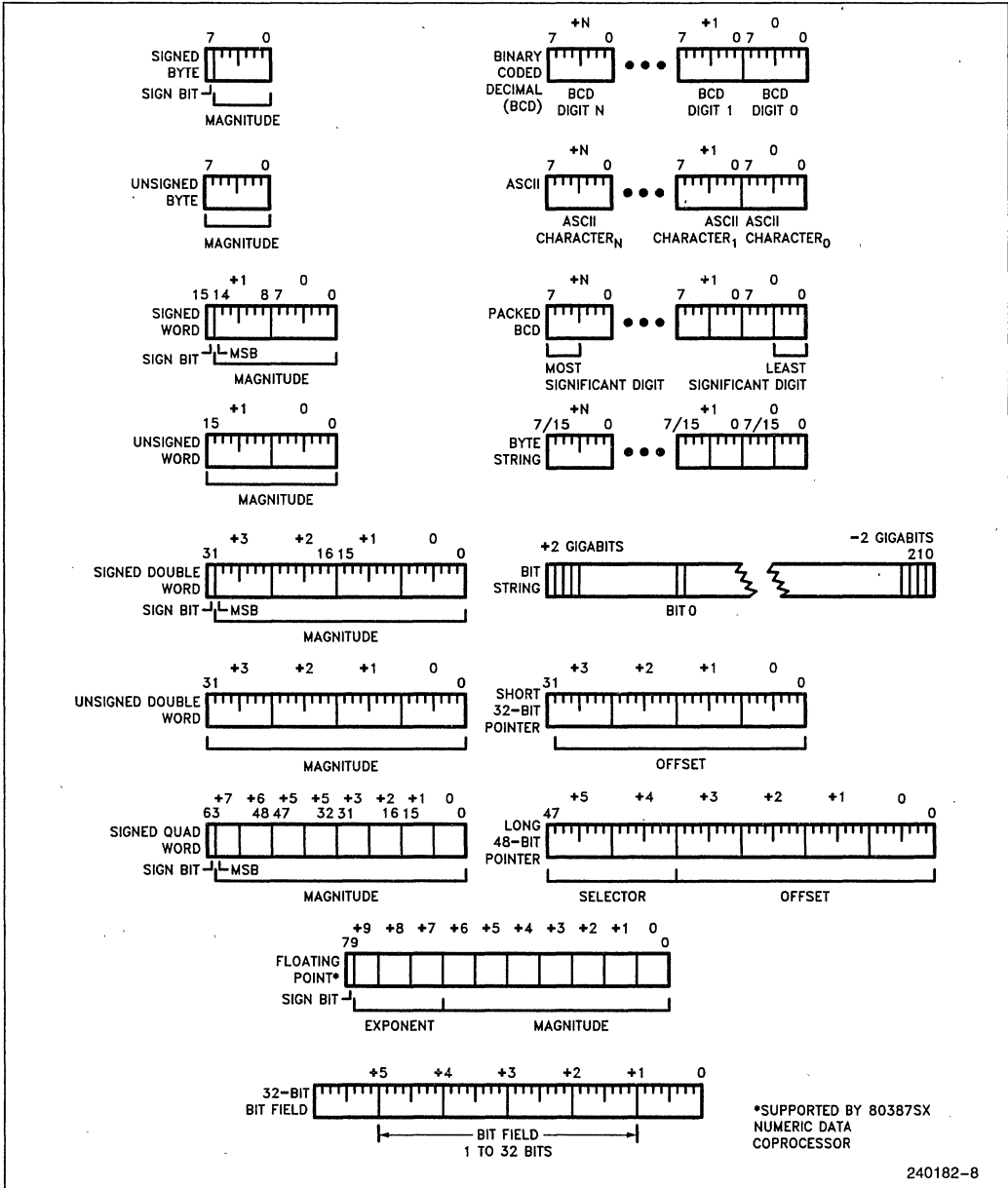
The 80376 supports all of the data types commonly used in high level languages:

- Bit: A single bit quantity.
- Bit Field: A group of up to 32 contiguous bits, which spans a maximum of four bytes.
- Bit String: A set of contiguous bits, on the 80376 bit strings can be up to 16 Mbits long.
- Byte: A signed 8-bit quantity.
- Unsigned Byte: An unsigned 8-bit quantity.
- Integer (Word): A signed 16-bit quantity.
- Long Integer (Double Word): A signed 32-bit quantity. All operations assume a 2's complement representation.
- Unsigned Integer (Word): An unsigned 16-bit quantity.
- Unsigned Long Integer (Double Word): An unsigned 32-bit quantity.
- Signed Quad Word: A signed 64-bit quantity.
- Unsigned Quad Word: An unsigned 64-bit quantity.
- Pointer: A 16- or 32-bit offset only quantity which indirectly references another memory location.
- Long Pointer: A full pointer which consists of a 16-bit segment selector and either a 16- or 32-bit offset.
- Char: A byte representation of an ASCII Alphanumeric or control character.
- String: A contiguous sequence of bytes, words or dwords. A string may contain between 1 byte and 16 Mbytes.
- BCD: A byte (unpacked) representation of decimal digits 0–9.
- Packed BCD: A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble.

When the 80376 is coupled with a numerics Coprocessor such as the 80387SX then the following common Floating Point types are supported.

Floating Point: A signed 32-, 64- or 80-bit real number representation. Floating point numbers are supported by the 80387SX numerics coprocessor.

Figure 2.5 illustrates the data types supported by the 80376 processor and the 80387SX coprocessor.



240182-8

Figure 2.5. 80376 Supported Data Types

2.6 I/O Space

The 80376 has two distinct physical address spaces: physical memory and I/O. Generally, peripherals are placed in I/O space although the 80376 also supports memory-mapped peripherals. The I/O space consists of 64 Kbytes which can be divided into 64K 8-bit ports, 32K 16-bit ports, or any combination of ports which add to no more than 64 Kbytes. The M/IO# pin acts as an additional address line, thus allowing the system designer to easily determine which address space the processor is accessing. Note that the I/O address refers to a physical address.

The I/O ports are accessed by the IN and OUT instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8-bit and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/IO# pin to be driven LOW. I/O port addresses 00F8H through 00FFH are reserved for use by Intel.

2.7 Interrupts and Exceptions

Interrupts and exceptions alter the normal program flow in order to handle external events, report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT N instruction, the processor treats software interrupts as exceptions.

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately **after** the interrupted instruction.

Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is supported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined. Thus, when an interrupt service routine has been completed, execution proceeds from the in-

struction immediately following the interrupted instruction. On the other hand the return address from an exception/fault routine will always point at the instruction causing the exception and include any leading instruction prefixes. Table 2.5 summarizes the possible interrupts for the 80376 and shows where the return address points to.

The 80376 has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. The interrupt vectors are 8-byte quantities, which are put in an Interrupt Descriptor Table. Of the 256 possible interrupts, 32 are reserved for use by Intel and the remaining 224 are free to be used by the system designer.

INTERRUPT PROCESSING

When an interrupt occurs the following actions happen. First, the current program address and the Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the 80376 which identifies the appropriate entry in the interrupt table. The table contains either an Interrupt Gate, a Trap Gate or a Task Gate that will point to an interrupt procedure or task. The user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the 80376 in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

Maskable Interrupt

Maskable interrupts are the most common way to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled HIGH and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions (string instructions have an "interrupt window" between memory moves which allows interrupts during long string moves). When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt (one of 224 user defined interrupts).

Table 2.5. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	Yes	FAULT
Debug Exception	1	Any Instruction	Yes	TRAP*
NMI Interrupt	2	INT 2 or NMI	No	NMI
One-Byte Interrupt	3	INT	No	TRAP
Interrupt on Overflow	4	INTO	No	TRAP
Array Bounds Check	5	BOUND	Yes	FAULT
Invalid OP-Code	6	Any Illegal Instruction	Yes	FAULT
Device Not Available	7	ESC, WAIT	Yes	FAULT
Double Fault	8	Any Instruction That Can Generate an Exception		ABORT
Coprocessor Segment Overrun	9	ESC	No	ABORT
Invalid TSS	10	JMP, CALL, IRET, INT	Yes	FAULT
Segment Not Present	11	Segment Register Instructions	Yes	FAULT
Stack Fault	12	Stack References	Yes	FAULT
General Protection Fault	13	Any Memory Reference	Yes	FAULT
Intel Reserved	14–15	—	—	—
Coprocessor Error	16	ESC, WAIT	Yes	FAULT
Intel Reserved	17–32			
Two-Byte Interrupt	0–255	INT n	No	TRAP

*Some debug exceptions may report both traps on the previous instruction, and faults on the next instruction.

Interrupts through Interrupt Gates automatically reset IF, disabling INTR requests. Interrupts through Trap Gates leave the state of the IF bit unchanged. Interrupts through a Task Gate change the IF bit according to the image of the EFLAGS register in the task's Task State Segment (TSS). When an IRET instruction is executed, the original state of the IF bit is restored.

Non-Maskable Interrupt

Non-maskable interrupts provide a method of servicing very high priority interrupts. When the NMI input is pulled HIGH it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt no interrupt acknowledgement sequence is performed for an NMI.

While executing the NMI servicing procedure, the 80376 will not service any further NMI request, or INT requests, until an interrupt return (IRET) instruc-

tion is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The disabling of INTR requests depends on the gate in IDT location 2.

Software Interrupts

A third type of interrupt/exception for the 80376 is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table.

A special case of the two byte software interrupt INT n is the one byte INT 3, or breakpoint interrupt. By inserting this one byte instruction in a program, the user can set breakpoints in his program as a debugging tool.

A final type of software interrupt, is the single step interrupt. It is discussed in **Single-Step Trap** (page 22).

INTERRUPT AND EXCEPTION PRIORITIES

Interrupts are externally-generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are **both** recognized at the **same** instruction boundary, the 80376 invokes the NMI service routine first. If, after the NMI service routine has been invoked, maskable interrupts are still enabled, then the 80376 will invoke the appropriate interrupt service routine.

As the 80376 executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2.6. This cycle is repeated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

INSTRUCTION RESTART

The 80376 fully supports restarting all instructions after faults. If an exception is detected in the instruction to be executed (exception categories 4 through 9 in Table 2.6), the 80376 device invokes the appropriate exception service routine. The 80376 is in a state that permits restart of the instruction.

DOUBLE FAULT

A Double fault (exception 8) results when the processor attempts to invoke an exception service routine for the segment exceptions (10, 11, 12 or 13), but in the process of doing so, detects an exception.

2.8 Reset and Initialization

When the processor is Reset the registers have the values shown in Table 2.7. The 80376 will then start executing instructions near the top of physical memory, at location 0FFFFF0H. A short JMP should be executed within the segment defined for power-up (see Table 2.7). The GDT should then be initialized for a start-up data and code segment followed by a far JMP that will load the segment descriptor cache with the new descriptor values. The IDT table, after reset, is located at physical address 0H, with a limit of 256 entries.

RESET forces the 80376 to terminate all execution and local bus activity. No instruction execution or bus activity will occur as long as Reset is active. Between 350 and 450 CLK2 periods after Reset becomes inactive, the 80376 will start executing instructions at the top of physical memory.

Table 2.6. Sequence of Exception Checking

Consider the case of the 80376 having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for external NMI and INTR.
3. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Faults decoding the next instruction (exception 6 if illegal opcode; or exception 13 if instruction is longer than 15 bytes, or privilege violation (i.e. not at IOPL or at CPL = 0).
6. If WAIT opcode, check if TS = 1 and MP = 1 (exception 7 if both are 1).
7. If ESCape opcode for numeric coprocessor, check if EM = 1 or TS = 1 (exception 7 if either are 1).
8. If WAIT opcode or ESCape opcode for numeric coprocessor, check ERROR# input signal (exception 16 if ERROR# input is asserted).
9. Check for Segmentation Faults that prevent transferring the entire memory quantity (exceptions 11, 12, 13).

Table 2.7. Register Values after Reset

Flag Word (EFLAGS)	uuuu0002H	(Note 1)
Machine Status Word (CR0)	uuuuuu1H	(Note 2)
Instruction Pointer (EIP)	0000FFFF0H	
Code Segment (CS)	F000H	(Note 3)
Data Segment (DS)	0000H	(Note 4)
Stack Segment (SS)	0000H	
Extra Segment (ES)	0000H	(Note 4)
Extra Segment (FS)	0000H	
Extra Segment (GS)	0000H	
EAX Register	0000H	(Note 5)
EDX Register	Component and Stepping ID	(Note 6)
All Other Registers	Undefined	(Note 7)

NOTES:

1. EFLAG Register. The upper 14 bits of the EFLAGS register are undefined, all defined flag bits are zero.
2. CR0: The defined 4 bits in the CR0 is equal to 1H.
3. The Code Segment Register (CS) will have its Base Address set to 0FFFF0000H and Limit set to 0FFFFH.
4. The Data and Extra Segment Registers (DS and ES) will have their Base Address set to 000000000H and Limit set to 0FFFFH.
5. If self-test is selected, the EAX should contain a 0 value. If a value of 0 is not found the self-test has detected a flaw in the part.
6. EDX register always holds component and stepping identifier.
7. All unidentified bits are Intel Reserved and should not be used.

2.9 Initialization

Because the 80376 processor starts executing in protected mode, certain precautions need to be taken during initialization. Before any far jumps can take place the GDT and/or LDT tables need to be setup and their respective registers loaded. Before interrupts can be initialized the IDT table must be setup and the IDTR must be loaded. The example code is shown below:

```

; *****
;
; This is an example of startup code to put either an 80376,
; 80386SX or 80386 into flat mode. All of memory is treated as
; simple linear RAM. There are no interrupt routines. The
; Builder creates the GDT-alias and IDT-alias and places them,
; by default, in GDT[1] and GDT[2]. Other entries in the GDT
; are specified in the Build file. After initialization it jumps
; to a C startup routine. To use this template, change this jmp
; address to that of your code, or make the label of your code
; "c_startup".
;
; This code was assembled and built using version 1.2 of the
; Intel RLL utilities and Intel 386ASM assembler.
;
;           ***      This code was tested      ***
;
; *****

```

```

NAME FLAT                ; name of the object module

EXTRN    c_startup:near ; this is the label jumped to after init

pe_flag    equ 1
data_selc  equ 20h      ; assume code is GDT[3], data GDT[4]

INIT_CODE  SEGMENT ER PUBLIC USE32      ; Segment base at 0ffffff80h

PUBLIC GDT_DESC

gdt_desc   dq  ?

PUBLIC     START

start:
    cld                ; clear direction flag
    smsw bx            ; check for processor (80376) at reset
    test bl,1          ; use SMSW rather than MOV for speed
    jnz pestart
realstart
    db 66h              ; is an 80386 and in real mode
    mov eax,offset gdt_desc ; move address of the GDT descriptor into eax
    xor ebx,ebx         ; clear ebx
    mov bh,ah           ; load 8 bits of address into bh
    move bl,al          ; load 8 bits of address into bl
    db 67h
    db 66h              ; use the 32-bit form of LGDT to load
    lgdt cs:[ebx]       ; the 32-bits of address into the GDTR
    smsw ax             ; go into protected mode (set PE bit)
    or al,pe_flag
    lmsw ax
    jmp next            ; flush prefetch queue
pestart:
    mov ebx,offset gdt_desc
    xor eax,eax
    mov ax,bx           ; lower portion of address only
    lgdt cs:[eax]
    xor ebx,ebx         ; initialize data selectors
    mov bl,data_selc   ; GDT[3]
    mov ds,bx
    mov ss,bx
    mov es,bx
    mov fs,bx
    mov gs,bx
    jmp pejump
next:
    xor ebx,ebx         ; initialize data selectors
    mov bl,data_selc   ; GDT[3]
    mov ds,bx
    mov ss,bx
    mov es,bx
    mov fs,bx
    mov gs,bx
    db 66h              ; for the 80386, need to make a 32-bit jump
pejump:
    jmp far ptr c_startup ; but the 80376 is already 32-bit.

    org 70h            ; only if segment base is at 0ffffff80h
    jmp short start
INIT_CODE ENDS
END

```

This code should be linked into your application for boot loadable code. The following build file illustrates how this is accomplished.

```

FLAT; -- build program id

SEGMENT
*segments (dpl=0),           -- Give all user segments a DPL of 0.
  _phantom_code_ (dpl=0),    -- These two segments are created by
  _phantom_data_ (dpl=0),    -- the builder when the FLAT control is used.
  init_code (base=0ffffff80h); -- Put startup code at the reset vector area.

GATE
i13 (entry=13, dpl=0, trap), -- trap gate disables interrupts
i32 (entry=32, dpl=0, interrupt), -- interrupt gates doesn't

TABLE
-- create GDT

GDT (LOCATION = GDT_DESC,     -- In a buffer starting at GDT_DESC,
    -- BLD386 places the GDT base and
    -- GDT limit values. Buffer must be
    -- 6 bytes long. The base and limit
    -- values are places in this buffer
    -- as two bytes of limit plus
    -- four bytes of base in the format
    -- required for use by the LGDT
    -- instruction.
    ENTRY = (3:_phantom_code_, -- Explicitly place segment
             4:_phantom_data_, -- entries into the GDT.
             5:code32,
             6:data,
             7:init_code)
    );

TASK
MAIN_TASK
(
  DPL = 0,           -- Task privilege level is 0.
  DATA = DATA,    -- Points to a segment that
                    -- indicates initial DS value.
  CODE = main,      -- Entry point is main, which
                    -- must be a public id.

  STACKS = (DATA),  -- Segment id points to stack
                    -- segment. Sets the initial SS:ESP.
  NO INTENABLED,    -- Disable interrupts.
  PRESENT           -- Present bit in TSS set to 1.
);

MEMORY
(RANGE = (EPROM = ROM(0ffff8000h..0fffffffh),
          DRAM = RAM(0..0ffffh)),
ALLOCATE = (EPROM = (MAIN_TASK)));

END

asm386 flatsim.a38 debug
asm386 application.a38 debug
bnd386 application.obj,flatsim.obj nolo debug oj (application.bnd)
bld386 application.bnd bf (flatsim.bld) bl flat

```

Commands to assemble and build a boot-loadable application named "application.a38". The initialization code is called "flatsim.a38", and build file is called "application.bl".

2.10 Self-Test

The 80376, like the 80386, has the capability to perform a self-test. The self-test checks the function of all of the Control ROM and most of the non-random logic of the part. Approximately one-half of the 80376 can be tested during self-test.

Self-Test is initiated on the 80376 when the RESET pin transitions from HIGH to LOW, and the BUSY# pin is LOW. The self-test takes about 220 clocks, or approximately 33 ms with a 16 MHz 80376 processor. At the completion of self-test the processor performs reset and begins normal operation. The part has successfully passed self-test if the contents of the EAX register is zero. If the EAX register is not zero then the self-test has detected a flaw in the part. If self-test is not selected after reset, EAX may be non-zero after reset.

2.11 Debugging Support

The 80376 provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

1. The code execution breakpoint opcode (0CCH).
2. The single-step capability provided by the TF bit in the flag register, and
3. The code and data breakpoint capability provided by the Debug Registers DR0-3, DR6, and DR7.

BREAKPOINT INSTRUCTION

A single-byte software interrupt (Int 3) breakpoint instruction is available for use by software debuggers. The breakpoint opcode is 0CCh, and generates an exception 3 trap when executed.

DEBUG REGISTERS

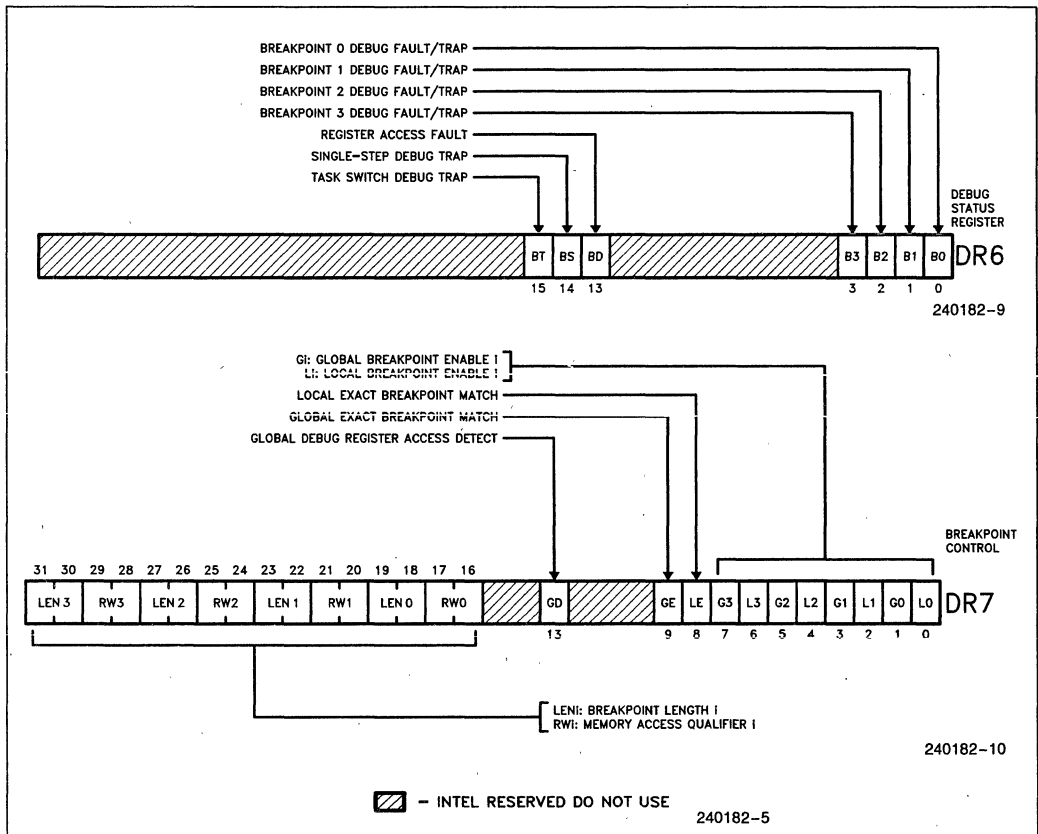


Figure 2.6. Debug Registers

SINGLE-STEP TRAP

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1.

The Debug Registers are an advanced debugging feature of the 80376. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT 3 breakpoint opcode.

The 80376 contains six Debug Registers, consisting of four breakpoint address registers and two breakpoint control registers. Initially after reset, breakpoints are in the disabled state; therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are auto-vectored to exception 1. Figure 2.6 shows the breakpoint status and control registers.

3.0 ARCHITECTURE

The Intel 80376 Embedded Processor has a physical address space of 16 Mbytes (2^{24} bytes) and allows the running of virtual memory programs of almost unlimited size (16 Kbytes \times 16 Mbytes or 256 Gbytes (2^{38} bytes)). In addition the 80376 provides a sophisticated memory management and a hardware-assisted protection mechanism.

3.1 Addressing Mechanism

The 80376 uses two components to form the logical address, a 16-bit selector which determines the linear base address of a segment, and a 32-bit effective address. The selector is used to specify an index into an operating system defined table (see Figure 3.1). The table contains the 32-bit base address of a given segment. The linear address is formed by adding the base address obtained from the table to the 32-bit effective address. This value is truncated to 24 bits to form the physical address, which is then placed on the address bus.

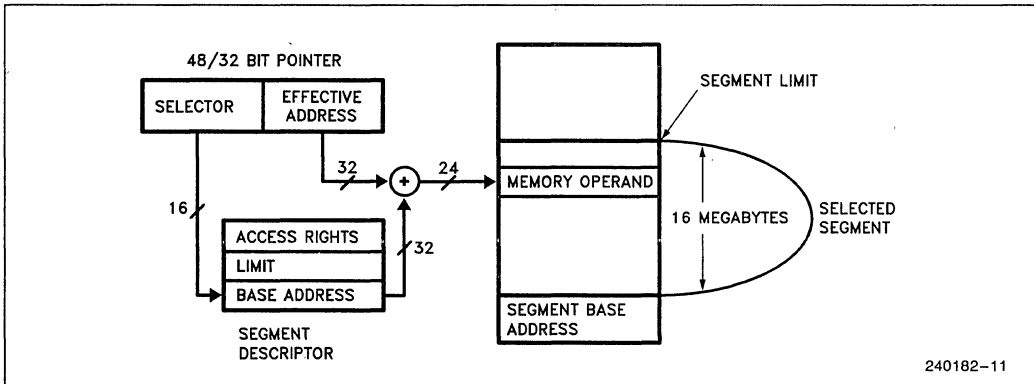


Figure 3.1. Address Calculation

3.2 Segmentation

Segmentation is one method of memory management and provides the basis for protection in the 80376. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about each segment, is stored in an 8-byte data structure called a descriptor. All of the descriptors in a system are contained in tables recognized by hardware.

TERMINOLOGY

The following terms are used throughout the discussion of descriptors, privilege levels and protection:

- PL: **Privilege Level**—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged.
- RPL: **Requestor Privilege Level**—The privilege level of the original supplier of the selector. RPL is determined by the least two significant bits of a selector.
- DPL: **Descriptor Privilege Level**—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.
- CPL: **Current Privilege Level**—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.
- EPL: **Effective Privilege Level**—The effective privilege level is the least privileged of the RPL and the DPL. EPL is the numerical maximum of RPL and DPL.
- Task: One instance of the execution of a program. Tasks are also referred to as processes.

DESCRIPTOR TABLES

The descriptor tables define all of the segments which are used in an 80376 system. There are three types of tables on the 80376 which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays, they can range in size between 8 bytes and 64 Kbytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address, and the 16-bit limit of each table.

Each of the tables have a register associated with it: GDTR, LDTR and IDTR; see Figure 3.2. The LGDT, LLDT and LIDT instructions load the base and limit of the Global, Local and Interrupt Descriptor Tables into the appropriate register. The SGDT, SLDT and SIDT store these base and limit values. These are privileged instructions.

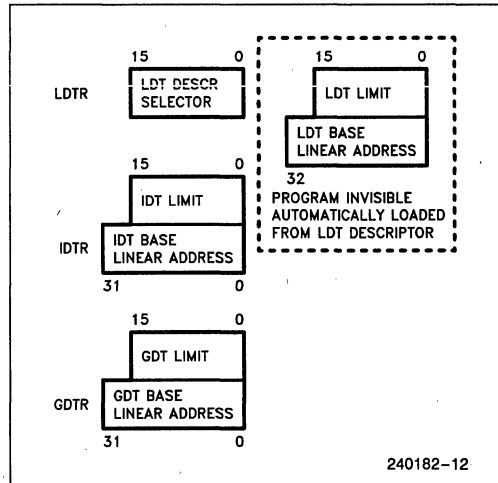


Figure 3.2. Descriptor Table Registers

Global Descriptor Table

The Global Descriptor Table (GDT) contains descriptors which are possibly available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for interrupt and trap descriptors. Every 80376 system contains a GDT. A simple 80376 system contains only 2 entries in the GDT; a code and a data descriptor. For maximum performance, descriptor tables should begin on even addresses.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

Local Descriptor Table

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This pro-

vides both isolation and protection for a task's segments, while still allowing global data to be shared among tasks.

Unlike the 6-byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT (see Figure 2.1).

INTERRUPT DESCRIPTOR TABLE

The third table needed for 80376 systems is the Interrupt Descriptor Table. The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced by INT instructions, external interrupt vectors, and exceptions.

DESCRIPTORS

The object to which the segment selector points to is called a descriptor. Descriptors are eight-byte quantities which contain attributes about a given region of linear address space. These attributes include the 32-bit logical base address of the seg-

ment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 3.3 shows the general format of a descriptor. All segments on the 80376 have three attribute fields in common: the Present bit (P), the Descriptor Privilege Level bits (DPL) and the Segment bit (S). P = 1 if the segment is loaded in physical memory, if P = 0 then any attempt to access the segment causes a not present exception (exception 11). The DPL is a two-bit field which specifies the protection level, 0-3, associated with a segment.

The 80376 has two main categories of segments: system segments, and non-system segments (for code and data). The segment bit, S, determines if a given segment is a system segment, a code segment or a data segment. If the S bit is 1 then the segment is either a code or data segment, if it is 0 then the segment is a system segment.

Note that although the 80376 is limited to a 16-Mbyte Physical address space (2²⁴), its base address allows a segment to be placed anywhere in a 4-Gbyte linear address space. When writing code for the 80376, users should keep code portability to an 80386 processor (or other processors with a larger physical address space) in mind. A segment base address can be placed anywhere in this 4-Gbyte linear address space, but a physical address will be

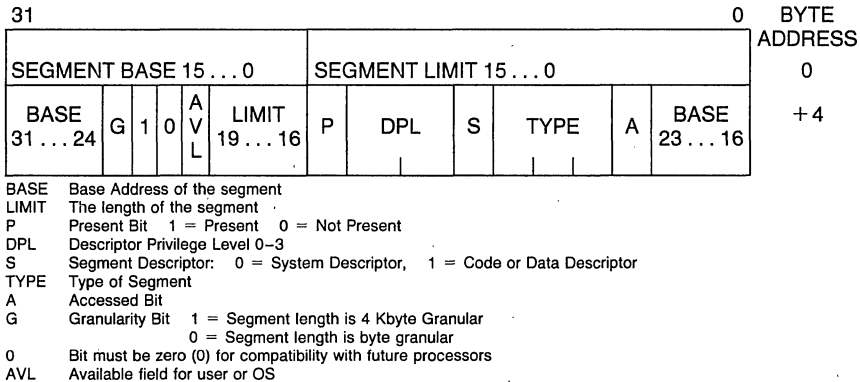


Figure 3.3. Segment Descriptors

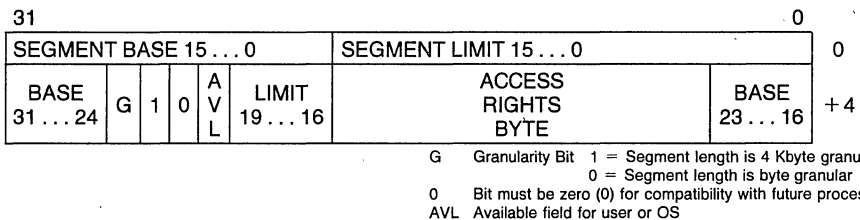


Figure 3.4. Code and Data Descriptors

Table 3.1. Access Rights Byte Definition for Code and Data Descriptors

Bit Position	Name	Function													
7	Present (P)	P = 1 Segment is mapped into physical memory. P = 0 No mapping to physical memory exits													
6-5	Descriptor Privilege Level (DPL)	Segment privilege attribute used in privilege tests.													
4	Segment Descriptor (S)	S = 1 Code or Data (includes stacks) segment descriptor S = 0 System Segment Descriptor or Gate Descriptor													
3 2 1	Executable (E) Expansion Direction (ED) Writable (W)	<table border="0"> <tr> <td>E = 0</td> <td>Descriptor type is data segment:</td> <td rowspan="4">} If Data Segment. (S = 1, E = 0)</td> </tr> <tr> <td>ED = 0</td> <td>Expand up segment, offsets must be ≤ limit.</td> </tr> <tr> <td>ED = 1</td> <td>Expand down segment, offsets must be > limit.</td> </tr> <tr> <td>W = 0</td> <td>Data segment may not be written into.</td> </tr> <tr> <td></td> <td>W = 1</td> <td>Data segment may be written into.</td> <td></td> </tr> </table>	E = 0	Descriptor type is data segment:	} If Data Segment. (S = 1, E = 0)	ED = 0	Expand up segment, offsets must be ≤ limit.	ED = 1	Expand down segment, offsets must be > limit.	W = 0	Data segment may not be written into.		W = 1	Data segment may be written into.	
E = 0	Descriptor type is data segment:	} If Data Segment. (S = 1, E = 0)													
ED = 0	Expand up segment, offsets must be ≤ limit.														
ED = 1	Expand down segment, offsets must be > limit.														
W = 0	Data segment may not be written into.														
	W = 1	Data segment may be written into.													
3 2 1	Executable (E) Conforming (C) Readable (R)	<table border="0"> <tr> <td>E = 1</td> <td>Descriptor type is code segment:</td> <td rowspan="4">} If Code Segment (S = 1, E = 1)</td> </tr> <tr> <td>C = 1</td> <td>Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.</td> </tr> <tr> <td>R = 0</td> <td>Code segment may not be read.</td> </tr> <tr> <td>R = 1</td> <td>Code segment may be read.</td> </tr> </table>	E = 1	Descriptor type is code segment:	} If Code Segment (S = 1, E = 1)	C = 1	Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.	R = 0	Code segment may not be read.	R = 1	Code segment may be read.				
E = 1	Descriptor type is code segment:	} If Code Segment (S = 1, E = 1)													
C = 1	Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.														
R = 0	Code segment may not be read.														
R = 1	Code segment may be read.														
0	Accessed (A)	A = 0 Segment has not been accessed. A = 1 Segment selector has been loaded into segment register or used by selector test instructions.													

generated that is a truncated version of this linear address. Truncation will be to the maximum number of address bits. It is recommended to place EPROM at the highest physical address and DRAM at the lowest physical addresses.

Code and Data Descriptors (S = 1)

Figure 3.4 shows the general format of a code and data descriptor and Table 3.1 illustrates how the bits in the Access Right Byte are interpreted.

Code and data segments have several descriptor fields in common. The accessed bit, A, is set whenever the processor accesses a descriptor. The granularity bit, G, specifies if a segment length is 1-byte-granular or 4-Kbyte-granular. Base address bits 31-24, which are normally found in 80386 descriptors, are not made externally available on the 80376. They do not affect the operation of the 80376. The A₃₁-A₂₄ field should be set to allow an 80386 to correctly execute with EPROM at the upper 4096 Mbytes of physical memory.

System Descriptor Formats (S = 0)

System segments describe information about operating system tables, tasks, and gates. Figure 3.5 shows the general format of system segment descriptors, and the various types of system segments.

80376 system descriptors (which are the same as 80386 descriptor types 2, 5, 9, B, C, E and F) contain a 32-bit logical base address and a 20-bit segment limit.

Selector Fields

A selector has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 3.6. The TI bit selects either the Global Descriptor Table or the Local Descriptor Table. The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

Segment Descriptor Cache

In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Since descriptor caches only change when a segment register is changed, programs which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

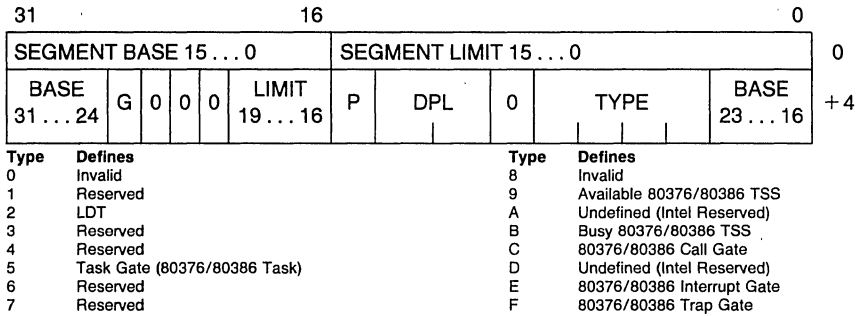


Figure 3.5. System Descriptors

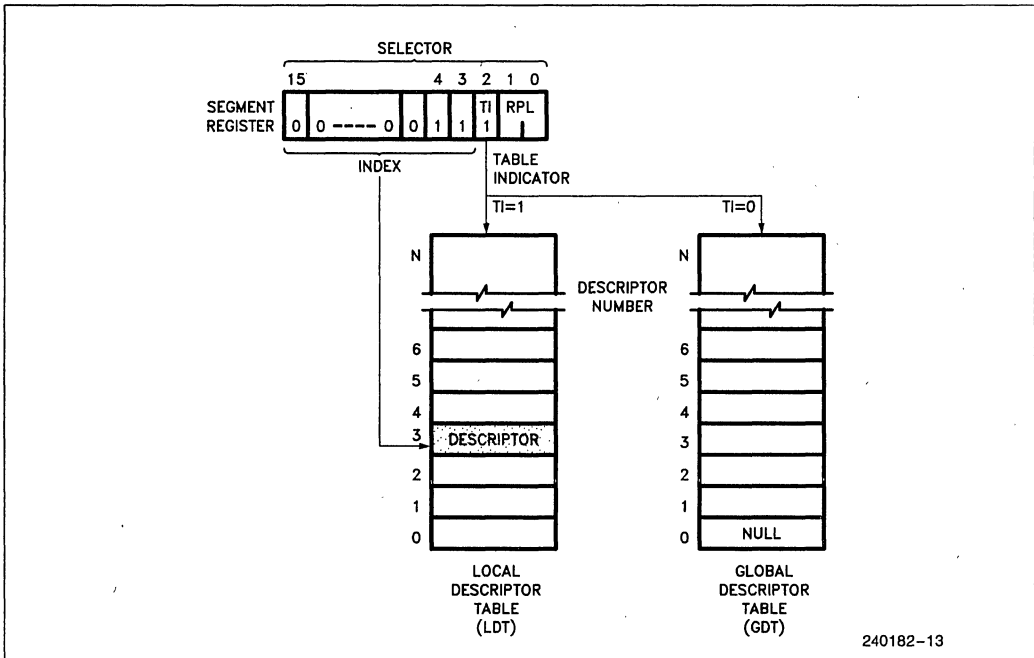


Figure 3.6. Example Descriptor Selection

3.3 Protection

The 80376 offers extensive protection features. These protection features are particularly useful in sophisticated embedded applications which use multitasking real-time operating systems. For simpler embedded applications these protection capabilities can be easily bypassed by making all applications run at privilege level (PL) 0.

—Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p**.

—A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p**.

RULES OF PRIVILEGE

The 80376 controls access to both data and procedures between levels of a task, according to the following rules.

PRIVILEGE LEVELS

At any point in time, a task on the 80376 always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies what the task's privilege level is. A task's CPL may only be changed

by control transfers through gate descriptors to a code segment with a different privilege level. Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level of the task for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (numerically larger) level of a task's CPL and a selector's RPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

I/O Privilege

The I/O privilege level (IOPL) lets the operating system code executing at CPL = 0 define the least privileged level at which I/O instructions can be used. An exception 13 (General Protection Violation) is generated if an I/O instruction is attempted when the CPL of the task is less privileged than the IOPL. The IOPL is stored in bits 13 and 14 of the EFLAGS register. The following instructions cause an exception 13 if the CPL is greater than IOPL: IN, INS, OUT, OUTS, STI, CLI and LOCK prefix.

Descriptor Access

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL as described above.

Any time an instruction loads a data segment register (DS, ES, FS, GS) the 80376 makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segment or readable code segments.

Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL, an exception 13 (general protection fault) is generated.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data segments. The DPL and RPL must equal the CPL of all other descriptor types or a privilege level violation will cause an exception 13. A stack not present fault causes an exception 12.

PRIVILEGE LEVEL TRANSFERS

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 3.2. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only by control transfers, using gates, task switches, and interrupt or trap gates.

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13.

CALL GATES

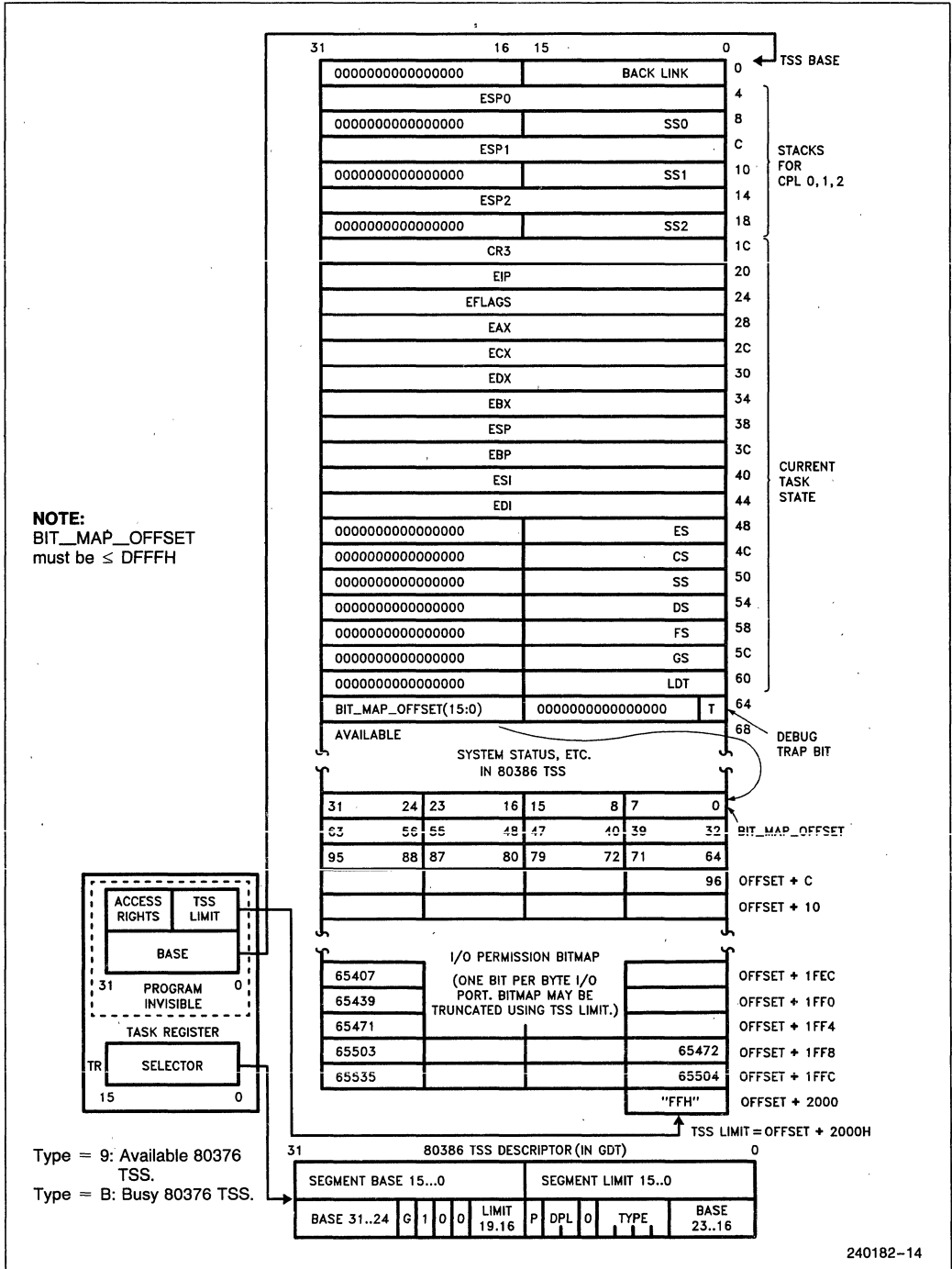
Gates provide protected indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures.

Table 3.2. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag register) = 0

**NT (Nested Task bit of flag register) = 1



TASK SWITCHING

A very important attribute of any multi-tasking operating system is its ability to rapidly switch between tasks or processes. The 80376 directly supports this operation by providing a task switch instruction in hardware. The 80376 task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task. Like transfer of control by gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot. For simple applications, the TSS and task switching may not be used. The TSS or task switch will not be used or occur if no task gates are present in the GDT, LDT or IDT.

The TSS descriptor points to a segment (see Figure 3.7) containing the entire 80376 execution state. A task gate descriptor contains a TSS selector. The limit of an 80376 TSS must be greater than 64H, and can be as large as 16 Mbytes. In the additional TSS space, the operating system is free to store additional information as the reason the task is inactive, the time the task has spent running, and open files belonging to the task. For maximum performance, TSS should start on an even address.

Each Task must have a TSS associated with it. The current TSS is identified by a special register in the 80376 called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with the TSS descriptor is loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was

interrupted. The current executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and CR0 register give information about the state of a task which is useful to the operating system. The Nested Task bit, NT, controls the function of the IRET instruction. If NT = 0 the IRET instruction performs the regular return. If NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT (The NT bit will be restored after execution of the interrupt handler). NT may also be set or cleared by POPF or IRET instructions.

The 80376 task state segment is marked busy by changing the descriptor type field from TYPE 9 to TYPE 0BH. Use of a selector that references a busy task state segment causes an exception 13.

The coprocessor's state is not automatically saved when a task switch occurs. The Task Switched Bit, TS, in the CR0 register helps deal with the coprocessor's state in a multi-tasking environment. Whenever the 80376 switches tasks, it sets the TS bit. The 80376 detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the coprocessor.

The T bit in the 80376 TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1 then upon entry to a new task a debug exception 1 will be generated.

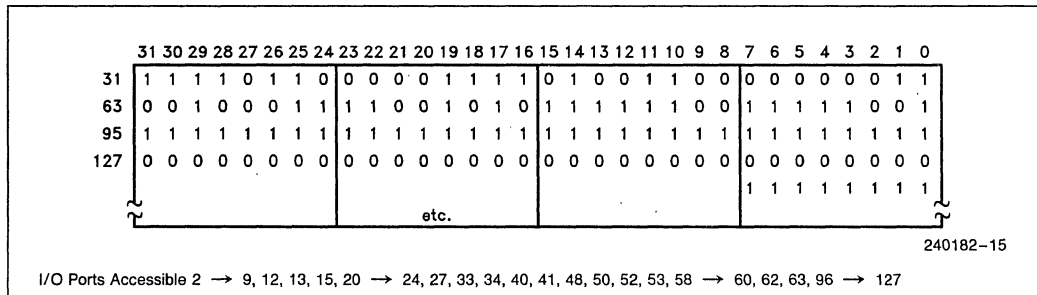


Figure 3.8. Sample I/O Permission Bit Map

PROTECTION AND I/O PERMISSION BIT MAP

The I/O instructions that directly refer to addresses in the processor's I/O space are IN, INS, OUT and OUTS. The 80376 has the ability to selectively trap references to specific I/O addresses. The structure that enables selective trapping is the *I/O Permission Bit Map* in the TSS segment (see Figures 3.7 and 3.8). The I/O permission map is a bit vector. The size of the map and its location in the TSS segment are variable. The processor locates the I/O permission map by means of the *I/O map base* field in the fixed portion of the TSS. The *I/O map base* field is 16 bits wide and contains the offset of the beginning of the I/O permission map.

If an I/O instruction (IN, INS, OUT or OUTS) is encountered, the processor first checks whether $CPL \leq IOPL$. If this condition is true, the I/O operation may proceed. If not true, the processor checks the I/O permission map.

Each bit in the map corresponds to an I/O port byte address; for example, the bit for port 41 is found at *I/O map base* + 5 linearly, ($5 \times 8 = 40$), bit offset 1. The processor tests all the bits that correspond to the I/O addresses spanned by an I/O operation; for example, a double word operation tests four bits corresponding to four adjacent byte addresses. If any tested bit is set, the processor signals a general protection exception. If all the tested bits are zero, the I/O operations may proceed.

It is not necessary for the I/O permission map to represent all the I/O addresses. I/O addresses not spanned by the map are treated as if they had one-bits in the map. The *I/O map base* should be at least one byte less than the TSS limit and the last byte beyond the I/O mapping information must contain all 1's.

Because the I/O permission map is in the TSS segment, different tasks can have different maps. Thus, the operating system can allocate ports to a task by changing the I/O permission map in the task's TSS.

IMPORTANT IMPLEMENTATION NOTE:

Beyond the last byte of I/O mapping information in the I/O permission bit map **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the 80376's TSS segment (see Figure 3.7).

4.0 FUNCTIONAL DATA

The Intel 80376 embedded processor features a straightforward functional interface to the external hardware. The 80376 has separate parallel buses for data and address. The data bus is 16 bits in width, and bidirectional. The address bus outputs 24-bit address values using 23 address lines and two-byte enable signals.

The 80376 has two selectable address bus cycles: pipelined and non-pipelined. The pipelining option allows as much time as possible for data access by

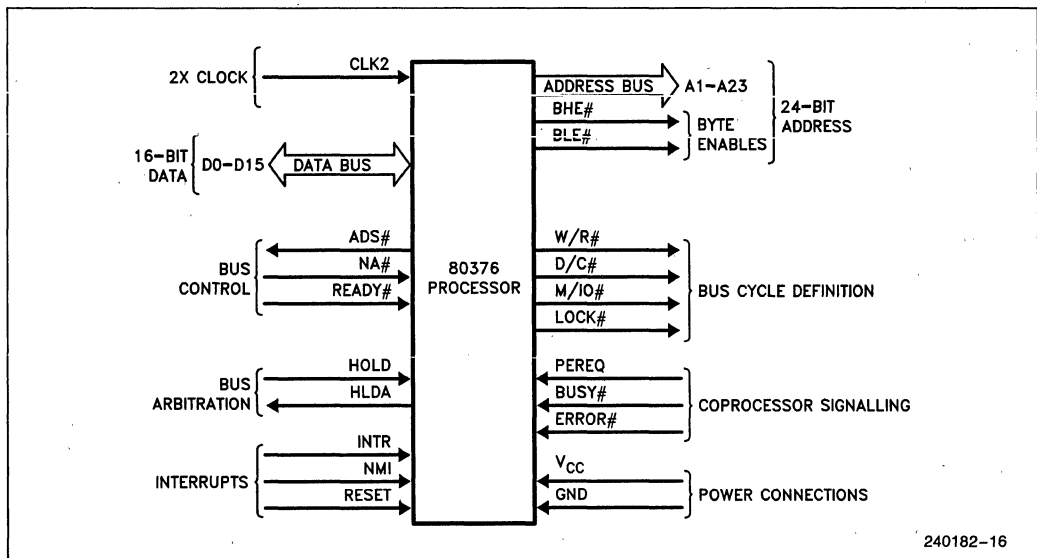


Figure 4.1. Functional Signal Groups

starting the pending bus cycle before the present bus cycle is finished. A non-pipelined bus cycle gives the highest bus performance by executing every bus cycle in two processor clock cycles. For maximum design flexibility, the address pipelining option is selectable on a cycle-by-cycle basis.

The processor's bus cycle is the basic mechanism for information transfer, either from system to processor, or from processor to system. 80376 bus cycles perform data transfer in a minimum of only two clock periods. On a 16-bit data bus, the maximum 80376 transfer bandwidth at 16 MHz is therefore 16 Mbytes/sec. However, any bus cycle will be extended for more than two clock periods if external hardware withholds acknowledgement of the cycle.

The 80376 can relinquish control of its local buses to allow mastership by other devices, such as direct memory access (DMA) channels. When relinquished, HLDA is the only output pin driven by the 80376, providing near-complete isolation of the processor from its system (all other output pins are in a float condition).

4.1 Signal Description Overview

Ahead is a brief description of the 80376 input and output signals arranged by functional groups. Note the # symbol at the end of a signal name indicates the active, or asserted, state occurs when the signal is at a LOW voltage. When no # is present after the signal name, the signal is asserted when at the HIGH voltage level.

Example signal: M/IO#—HIGH voltage indicates Memory selected

—LOW voltage indicates I/O selected

The signal descriptions sometimes refer to A.C. timing parameters, such as "t₂₅ Reset Setup Time" and "t₂₆ Reset Hold Time." The values of these parameters can be found in Table 6.4.

CLOCK (CLK2)

CLK2 provides the fundamental timing for the 80376. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock is comprised of two

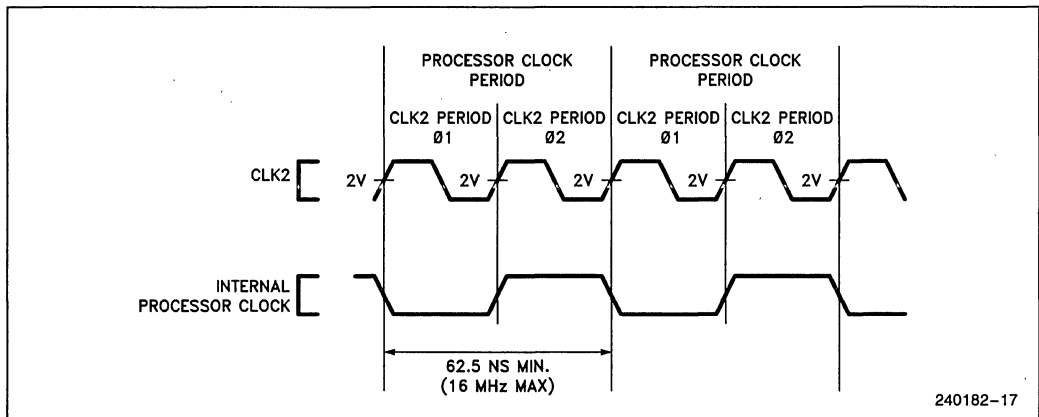


Figure 4.2. CLK2 Signal and Internal Processor Clock

phases, "phase one" and "phase two". Each CLK2 period is a phase of the internal clock. Figure 4.2 illustrates the relationship. If desired, the phase of the internal processor clock can be synchronized to a known phase by ensuring the falling edge of the RESET signal meets the applicable setup and hold times t_{25} and t_{26} .

DATA BUS (D₁₅-D₀)

These three-state bidirectional signals provide the general purpose data path between the 80376 and other devices. The data bus outputs are active HIGH and will float during bus hold acknowledge. Data bus reads require that read-data setup and hold times t_{21} and t_{22} be met relative to CLK2 for correct operation.

ADDRESS BUS (BHE#, BLE#, A₂₃-A₁)

These three-state outputs provide physical memory addresses or I/O port addresses. A₂₃-A₁₆ are LOW during I/O transfers except for I/O transfers automatically generated by coprocessor instructions.

During coprocessor I/O transfers, A₂₂-A₁₆ are driven LOW, and A₂₃ is driven HIGH so that this address line can be used by external logic to generate the coprocessor select signal. Thus, the I/O address driven by the 80376 for coprocessor commands is 8000F8H, and the I/O address driven by the 80376 processor for coprocessor data is 8000FCH or 8000FEH.

The address bus is capable of addressing 16 Mbytes of physical memory space (000000H through 0FFFFFFH), and 64 Kbytes of I/O address space (000000H through 00FFFFH) for programmed I/O. The address bus is active HIGH and will float during bus hold acknowledge.

The Byte Enable outputs BHE# and BLE# directly indicate which bytes of the 16-bit data bus are involved with the current transfer. BHE# applies to D₁₅-D₈ and BLE# applies to D₇-D₀. If both BHE# and BLE# are asserted, then 16 bits of data are being transferred. See Table 4.1 for a complete decoding of these signals. The byte enables are active LOW and will float during bus hold acknowledge.

Table 4.1. Byte Enable Definitions

BHE #	BLE #	Function
0	0	Word Transfer
0	1	Byte Transfer on Upper Byte of the Data Bus, D ₁₅ -D ₈
1	0	Byte Transfer on Lower Byte of the Data Bus, D ₇ -D ₀
1	1	Never Occurs

**BUS CYCLE DEFINITION SIGNALS
(W/R#, D/C#, M/IO#, LOCK#)**

These three-state outputs define the type of bus cycle being performed: W/R# distinguishes between write and read cycles, D/C# distinguishes between data and control cycles, M/IO# distinguishes between memory and I/O cycles, and LOCK# distinguishes between locked and unlocked bus cycles. All of these signals are active LOW and will float during bus acknowledge.

The primary bus cycle definition signals are W/R#, D/C# and M/IO#, since these are the signals driven valid as ADS# (Address Status output) becomes active. The LOCK# signal is driven valid at the same time the bus cycle begins, which due to address pipelining, could be after ADS# becomes active. Exact bus cycle definitions, as a function of W/R#, D/C# and M/IO# are given in Table 4.2.

LOCK# indicates that other system bus masters are not to gain control of the system bus while it is active. LOCK# is activated on the CLK2 edge that begins the first locked bus cycle (i.e., it is not active at the same time as the other bus cycle definition pins) and is deactivated when ready is returned to the end of the last bus cycle which is to be locked. The beginning of a bus cycle is determined when READY# is returned in a previous bus cycle and another is pending (ADS# is active) or the clock in which ADS# is driven active if the bus was idle. This means that it follows more closely with the write data rules when it is valid, but may cause the bus to be locked longer than desired. The LOCK# signal may be explicitly activated by the LOCK prefix on certain instructions. LOCK# is always asserted when executing the XCHG instruction, during descriptor updates, and during the interrupt acknowledge sequence.

**BUS CONTROL SIGNALS
(ADS#, READY#, NA#)**

The following signals allow the processor to indicate when a bus cycle has begun, and allow other system hardware to control address pipelining and bus cycle termination.

Address Status (ADS#)

This three-state output indicates that a valid bus cycle definition and address (W/R#, D/C#, M/IO#, BHE#, BLE# and A₂₃-A₁) are being driven at the 80376 pins. ADS# is an active LOW output. Once ADS# is driven active, valid address, byte enables, and definition signals will not change. In addition, ADS# will remain active until its associated bus cycle begins (when READY# is returned for the previous bus cycle when running pipelined bus cycles). ADS# will float during bus hold acknowledge. See sections **Non-Pipelined Bus Cycles** (page 43) and **Pipelined Bus Cycles** (page 45) for additional information on how ADS# is asserted for different bus states.

Transfer Acknowledge (READY#)

This input indicates the current bus cycle is complete, and the active bytes indicated by BHE# and BLE# are accepted or provided. When READY# is sampled active during a read cycle or interrupt acknowledge cycle, the 80376 latches the input data and terminates the cycle. When READY# is sampled active during a write cycle, the processor terminates the bus cycle.

Table 4.2. Bus Cycle Definition

M/IO#	D/C#	W/R#	Bus Cycle Type	Locked?
0	0	0	INTERRUPT ACKNOWLEDGE	Yes
0	0	1	Does Not Occur	—
0	1	0	I/O DATA READ	No
0	1	1	I/O DATA WRITE	No
1	0	0	MEMORY CODE READ	No
1	0	1	HALT: SHUTDOWN: Address = 2 Address = 0 BHE# = 1 BHE# = 1 BLE# = 0 BLE# = 0	No
1	1	0	MEMORY DATA READ	Some Cycles
1	1	1	MEMORY DATA WRITE	Some Cycles

READY# is ignored on the first bus state of all bus cycles, and sampled each bus state thereafter until asserted. READY# must eventually be asserted to acknowledge every bus cycle, including Halt Indication and Shutdown Indication bus cycles. When being sampled, READY# must always meet setup and hold times t_{19} and t_{20} for correct operation.

Next Address Request (NA#)

This is used to request pipelining. This input indicates the system is prepared to accept new values of BHE#, BLE#, A₂₃-A₁, W/R#, D/C# and M/IO# from the 80376 even if the end of the current cycle is not being acknowledged on READY#. If this input is active when sampled, the next bus cycle's address and status signals are driven onto the bus, provided the next bus request is already pending internally. NA# is ignored in clock cycles in which ADS# or READY# is activated. This signal is active LOW and must satisfy setup and hold times t_{15} and t_{16} for correct operation. See **Pipelined Bus Cycles** (page 45) and **Read and Write Cycles** (page 42) for additional information.

BUS ARBITRATION SIGNALS (HOLD, HLDA)

This section describes the mechanism by which the processor relinquishes control of its local buses when requested by another bus master device. See **Entering and Exiting Hold Acknowledge** (page 52) for additional information.

Bus Hold Request (HOLD)

This input indicates some device other than the 80376 requires bus mastership. When control is granted, the 80376 floats A₂₃-A₁, BHE#, BLE#, D₁₅-D₀, LOCK#, M/IO#, D/C#, W/R# and ADS#, and then activates HLDA, thus entering the bus hold acknowledge state. The local bus will remain granted to the requesting master until HOLD becomes inactive. When HOLD becomes inactive, the 80376 will deactivate HLDA and drive the local bus (at the same time), thus terminating the hold acknowledge condition.

HOLD must remain asserted as long as any other device is a local bus master. External pull-up resistors may be required when in the hold acknowledge state since none of the 80376 floated outputs have internal pull-up resistors. See **Resistor Recommendations** (page 59) for additional information. HOLD is not recognized while RESET is active but is recognized during the time between the high-to-low transition of RESET and the first instruction fetch. If RESET is asserted while HOLD is asserted, RESET has priority and places the bus into an idle state, rather than the hold acknowledge (high-impedance) state.

HOLD is a level-sensitive, active HIGH, synchronous input. HOLD signals must always meet setup and hold times t_{23} and t_{24} for correct operation.

Bus Hold Acknowledge (HLDA)

When active (HIGH), this output indicates the 80376 has relinquished control of its local bus in response to an asserted HOLD signal, and is in the bus Hold Acknowledge state.

The Bus Hold Acknowledge state offers near-complete signal isolation. In the Hold Acknowledge state, HLDA is the only signal being driven by the 80376. The other output signals or bidirectional signals (D₁₅-D₀, BHE#, BLE#, A₂₃-A₁, W/R#, D/C#, M/IO#, LOCK# and ADS#) are in a high-impedance state so the requesting bus master may control them. These pins remain OFF throughout the time that HLDA remains active (see Table 4.3). Pull-up resistors may be desired on several signals to avoid spurious activity when no bus master is driving them. See **Resistor Recommendations** (page 59) for additional information.

When the HOLD signal is made inactive, the 80376 will deactivate HLDA and drive the bus. One rising edge on the NMI input is remembered for processing after the HOLD input is negated.

Table 4.3. Output Pin State during HOLD

Pin Value	Pin Names
1	HLDA
Float	LOCK#, M/IO#, D/C#, W/R#, ADS#, A ₂₃ -A ₁ , BHE#, BLE#, D ₁₅ -D ₀

In addition to the normal usage of Hold Acknowledge with DMA controllers or master peripherals, the near-complete isolation has particular attractiveness during system test when test equipment drives the system, and in hardware-fault-tolerant applications.

Hold Latencies

The maximum possible HOLD latency depends on the software being executed. The actual HOLD latency at any time depends on the current bus activity, the state of the LOCK# signal (internal to the CPU) activated by the LOCK# prefix, and interrupts. The 80376 will not honor a HOLD request until the current bus operation is complete. Table 4.4 shows the types of bus operations that can affect HOLD latency, and indicates the types of delays that

these operations may introduce. When considering maximum HOLD latencies, designers must select which of these bus operations are possible, and then select the maximum latency form among them.

The 80376 breaks 32-bit data or I/O accesses into 2 internally locked 16-bit bus cycles; the LOCK# signal is not asserted. The 80376 breaks unaligned 16-bit or 32-bit data or I/O accesses into 2 or 3 internally locked 16-bit bus cycles. Again the LOCK# signal is not asserted but a HOLD request will not be recognized until the end of the entire transfer.

Wait states affect HOLD latency. The 80376 will not honor a HOLD request until the end of the current bus operation, no matter how many wait states are required. Systems with DMA where data transfer is critical must insure that READY# returns sufficiently soon.

COPROCESSOR INTERFACE SIGNALS (PEREQ, BUSY#, ERROR#)

In the following sections are descriptions of signals dedicated to the numeric coprocessor interface. In addition to the data bus, address bus, and bus cycle definition signals, these following signals control communication between the 80376 and the 80387SX processor extension.

Coprocessor Request (PEREQ)

When asserted (HIGH), this input signal indicates a coprocessor request for a data operand to be transferred to/from memory by the 80376. In response, the 80376 transfers information between the coprocessor and memory. Because the 80376 has internally stored the coprocessor opcode being executed, it performs the requested data transfer with the correct direction and memory address.

PEREQ is a level-sensitive active HIGH asynchronous signal. Setup and hold times, t_{29} and t_{30} , relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This signal is provided with a weak internal pull-down resistor of around 20 K Ω to ground so that it will not float active when left unconnected.

Coprocessor Busy (BUSY#)

When asserted (LOW), this input indicates the coprocessor is still executing an instruction, and is not yet able to accept another. When the 80376 encounters any coprocessor instruction which operates on the numerics stack (e.g. load, pop, or arithmetic operation), or the WAIT instruction, this input is first automatically sampled until it is seen to be inactive. This sampling of the BUSY# input prevents overrunning the execution of a previous coprocessor instruction.

The F(N)INIT, F(N)CLEX coprocessor instructions are allowed to execute even if BUSY# is active, since these instructions are used for coprocessor initialization and exception-clearing.

BUSY# is an active LOW, level-sensitive asynchronous signal. Setup and hold times, t_{29} and t_{30} , relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This pin is provided with a weak internal pull-up resistor of around 20 K Ω to V_{CC} so that it will not float active when left unconnected.

BUSY# serves an additional function. If BUSY# is sampled LOW at the falling edge of RESET, the 80376 processor performs an internal self-test (see **Bus Activity During and Following Reset** on page 54). If BUSY# is sampled HIGH, no self-test is performed.

Coprocessor Error (ERROR#)

When asserted (LOW), this input signal indicates that the previous coprocessor instruction generated a coprocessor error of a type not masked by the coprocessor's control register. This input is automatically sampled by the 80376 when a coprocessor instruction is encountered, and if active, the 80376 generates exception 16 to access the error-handling software.

Several coprocessor instructions, generally those which clear the numeric error flags in the coprocessor or save coprocessor state, do execute without the 80376 generating exception 16 even if ERROR# is active. These instructions are FNINIT, FNCLEX, FNSTSW, FNSTSWAX, FNSTCW, FNSTENV and FNSAVE.

ERROR# is an active LOW, level-sensitive asynchronous signal. Setup and hold times t_{29} and t_{30} , relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This pin is provided with a weak internal pull-up resistor of around 20 K Ω to V_{CC} so that it will not float active when left unconnected.

INTERRUPT SIGNALS (INTR, NMI, RESET)

The following descriptions cover inputs that can interrupt or suspend execution of the processor's current instruction stream.

Maskable Interrupt Request (INTR)

When asserted, this input indicates a request for interrupt service, which can be masked by the 80376 Flag Register IF bit. When the 80376 responds to the INTR input, it performs two interrupt acknowledge bus cycles and, at the end of the second, latches an 8-bit interrupt vector on D7–D0 to identify the source of the interrupt.

INTR is an active HIGH, level-sensitive asynchronous signal. Setup and hold times, t_{27} and t_{28} , relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. To assure recognition of an INTR request, INTR should remain active until the first interrupt acknowledge bus cycle begins. INTR is sampled at the beginning of every instruction. In order to be recognized at a particular instruction boundary, INTR must be active at least eight CLK2 clock periods before the beginning of the execution of the instruction. If recognized, the 80376 will begin execution of the interrupt.

Non-Maskable Interrupt Request (NMI)

This input indicates a request for interrupt service which cannot be masked by software. The non-maskable interrupt request is always processed according to the pointer or gate in slot 2 of the interrupt table. Because of the fixed NMI slot assignment, no interrupt acknowledge cycles are performed when processing NMI.

NMI is an active HIGH, rising edge-sensitive asynchronous signal. Setup and hold times, t_{27} and t_{28} , relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. To assure recognition of NMI, it must be inactive for at least eight CLK2 periods, and then be active for at least eight CLK2 periods before the beginning of the execution of an instruction.

Once NMI processing has begun, no additional NMI's are processed until after the next IRET instruction, which is typically the end of the NMI serv-

ice routine. If NMI is re-asserted prior to that time, however, one rising edge on NMI will be remembered for processing after executing the next IRET instruction.

Interrupt Latency

The time that elapses before an interrupt request is serviced (interrupt latency) varies according to several factors. This delay must be taken into account by the interrupt source. Any of the following factors can affect interrupt latency:

1. If interrupts are masked, and INTR request will not be recognized until interrupts are reenabled.
2. If an NMI is currently being serviced, an incoming NMI request will not be recognized until the 80376 encounters the IRET instruction.
3. An interrupt request is recognized only on an instruction boundary of the 80376 *Execution Unit* except for the following cases:
 - Repeat string instructions can be interrupted after each iteration.
 - If the instruction loads the Stack Segment register, an interrupt is not processed until after the following instruction, which should be an ESP load. This allows the entire stack pointer to be loaded without interruption.
 - If an instruction sets the interrupt flag (enabling interrupts), an interrupt is not processed until after the next instruction.
4. Saving the Flags register and CS:EIP registers.
5. If interrupt service routine requires a task switch, time must be allowed for the task switch.
6. If the interrupt service routine saves registers that are not automatically saved by the 80376.

RESET

This input signal suspends any operation in progress and places the 80376 in a known reset state. The 80376 is reset by asserting RESET for 15 or more CLK2 periods (80 or more CLK2 periods before requesting self-test). When RESET is active, all other input pins are ignored, and all other bus pins are driven to an idle bus state as shown in Table 4.5. If RESET and HOLD are both active at a point in time, RESET takes priority even if the 80376 was in a Hold Acknowledge state prior to RESET active.

RESET is an active HIGH, level-sensitive synchronous signal. Setup and hold times, t_{25} and t_{26} , must be met in order to assure proper operation of the 80376.

Table 4.5. Pin State (Bus Idle) during RESET

Pin Name	Signal Level during RESET
ADS#	1
D ₁₅ -D ₀	Float
BHE#, BLE#	0
A ₂₃ -A ₁	1
W/R#	0
D/C#	1
M/IO#	0
LOCK#	1
HLDA	0

4.2 Bus Transfer Mechanism

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte and word lengths may be transferred without restrictions on physical address alignment. Any byte boundary may be used, although two physical bus cycles are performed as required for unaligned operand transfers.

The 80376 processor address signals are designed to simplify external system hardware. BHE# and BLE# provide linear selects for the two bytes of the 16-bit data bus.

Byte Enable outputs BHE# and BLE# are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 4.6.

Table 4.6. Byte Enables and Associated Data and Operand Bytes

Byte Enable	Associated Data Bus Signals
BHE#	D ₁₅ -D ₈ (Byte 1—Most Significant)
BLE#	D ₇ -D ₀ (Byte 0—Least Significant)

Each bus cycle is composed of at least two bus states. Each bus state requires one processor clock period. Additional bus states added to a single bus cycle are called wait states. See **Bus Functional Description** (page 39) for additional information.

4.3 Memory and I/O Spaces

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. As shown in Figure 4.3, physical memory addresses range from 000000H to 0FFFFFFH (16 Mbytes) and I/O addresses from 000000H to 00FFFFH (64 Kbytes). Note the I/O addresses used by the automatic I/O cycles for coprocessor communication are 8000F8H to 8000FFH, beyond the address range of programmed I/O, to allow easy generation of a coprocessor chip select signal using the A₂₃ and M/IO# signals.

OPERAND ALIGNMENT

With the flexibility of memory addressing on the 80376, it is possible to transfer a logical operand that spans more than one physical Dword or word of memory or I/O. Examples are 32-bit Dword or 16-bit word operands beginning at addresses not evenly divisible by 2.

Operand alignment and size dictate when multiple bus cycles are required. Table 4.6a describes the transfer cycles generated for all combinations of logical operand lengths and alignment.

Table 4.6a. Transfer Bus Cycles for Bytes, Words and Dwords

	Byte-Length of Logical Operand								
	1		2		4				
Physical Byte Address in Memory (Low-Order Bits)	xx	00	01	10	11	00	01	10	11
Transfer Cycles	b	w	lb, hb	w	hb, lb	lw, hw	hb, lb, mw	hw, lw	mw, hb, lb

Key: b = byte transfer
w = word transfer
l = low-order portion
m = mid-order portion
x = don't care
h = high-order portion

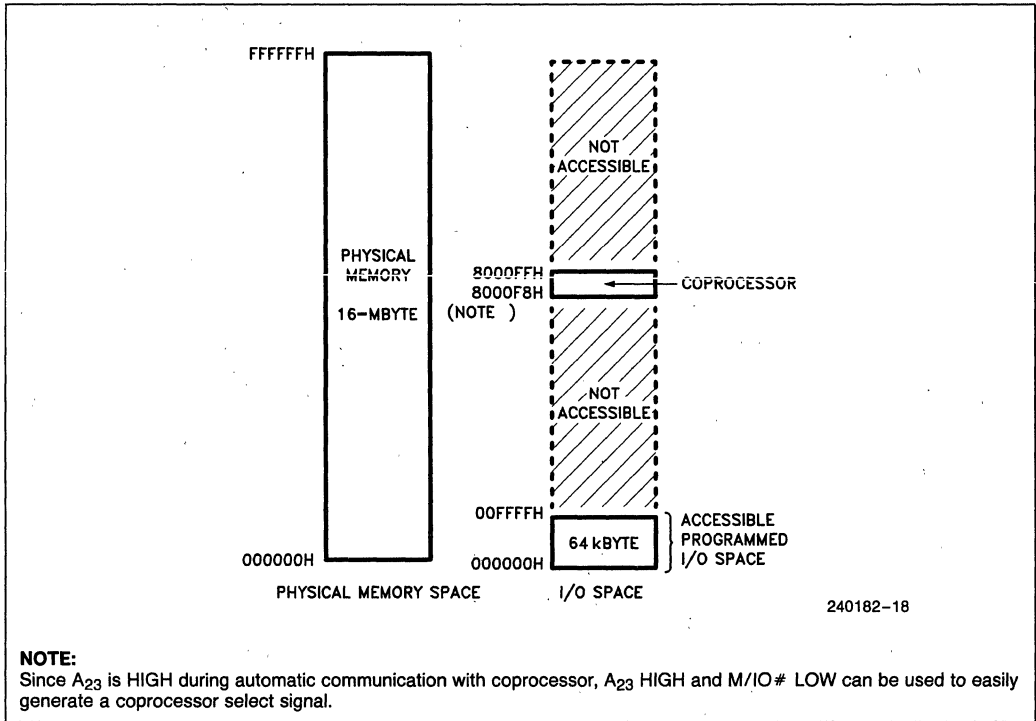


Figure 4.3. Physical Memory and I/O Spaces

4.4 Bus Functional Description

The 80376 has separate, parallel buses for data and address. The data bus is 16 bits in width, and bidirectional. The address bus provides a 24-bit value using 23 signals for the 23 upper-order address bits and 2 Byte Enable signals to directly indicate the active bytes. These buses are interpreted and controlled by several definition signals.

The definition of each bus cycle is given by three signals: $M/IO\#$, $W/R\#$ and $D/C\#$. At the same time, a valid address is present on the byte enable signals, $BHE\#$ and $BLE\#$, and the other address signals $A_{23}-A_1$. A status signal, $ADS\#$, indicates when the 80376 issues a new bus cycle definition and address.

Collectively, the address bus, data bus and all associated control signals are referred to simply as "the bus". When active, the bus performs one of the bus cycles below:

1. Read from memory space
2. Locked read from memory space
3. Write to memory space
4. Locked write to memory space

5. Read from I/O space (or coprocessor)
6. Write to I/O space (or coprocessor)
7. Interrupt acknowledge (always locked)
8. Indicate halt, or indicate shutdown

Table 4.2 shows the encoding of the bus cycle definition signals for each bus cycle. See **Bus Cycle Definition Signals** (page 35) for additional information.

When the 80376 bus is not performing one of the activities listed above, it is either Idle or in the Hold Acknowledge state, which may be detected by external circuitry. The idle state can be identified by the 80376 giving no further assertions on its address strobe output ($ADS\#$) since the beginning of its most recent bus cycle, and the most recent bus cycle having been terminated. The hold acknowledge state is identified by the 80376 asserting its hold acknowledge (HLDA) output.

The shortest time unit of bus activity is a bus state. A bus state is one processor clock period (two CLK2 periods) in duration. A complete data transfer occurs during a bus cycle, composed of two or more bus states.

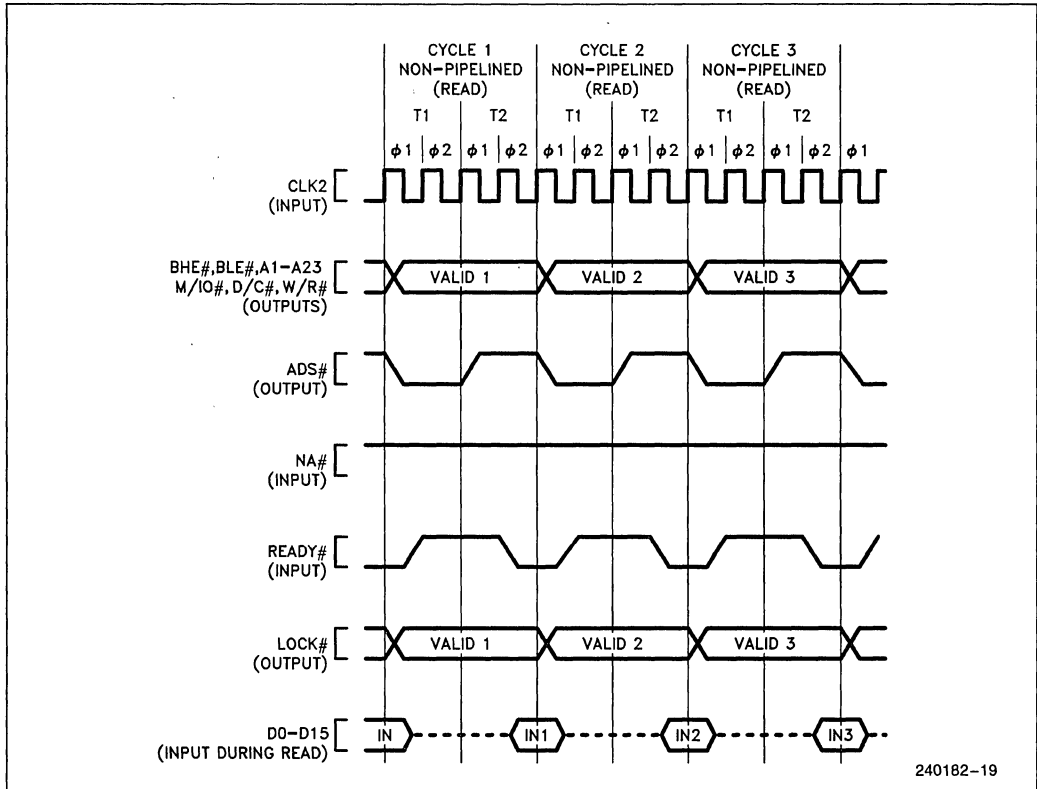


Figure 4.4. Fastest Read Cycles with Non-Pipelined Timing

The fastest 80376 bus cycle requires only two bus states. For example, three consecutive bus read cycles, each consisting of two bus states, are shown by Figure 4.4. The bus states in each cycle are named T1 and T2. Any memory or I/O address may be accessed by such a two-state bus cycle, if the external hardware is fast enough.

Every bus cycle continues until it is acknowledged by the external system hardware, using the 80376 READY# input. Acknowledging the bus cycle at the end of the first T2 results in the shortest bus cycle, requiring only T1 and T2. If READY# is not immediately asserted however, T2 states are repeated indefinitely until the READY# input is sampled active.

The pipelining option provides a choice of bus cycle timings. Pipelined or non-pipelined cycles are

selectable on a cycle-by-cycle basis with the Next Address (NA#) input.

When pipelining is selected the address (BHE#, BLE# and A₂₃-A₁) and definition (W/R#, D/C#, M/IO# and LOCK#) of the next cycle are available before the end of the current cycle. To signal their availability, the 80376 address status output (ADS#) is asserted. Figure 4.5 illustrates the fastest read cycles with pipelined timing.

Note from Figure 4.5 the fastest bus cycles using pipelining require only two bus states, named T1P and T2P. Therefore pipelined cycles allow the same data bandwidth as non-pipelined cycles, but address-to-data access time is increased by one T-state time compared to that of a non-pipelined cycle.

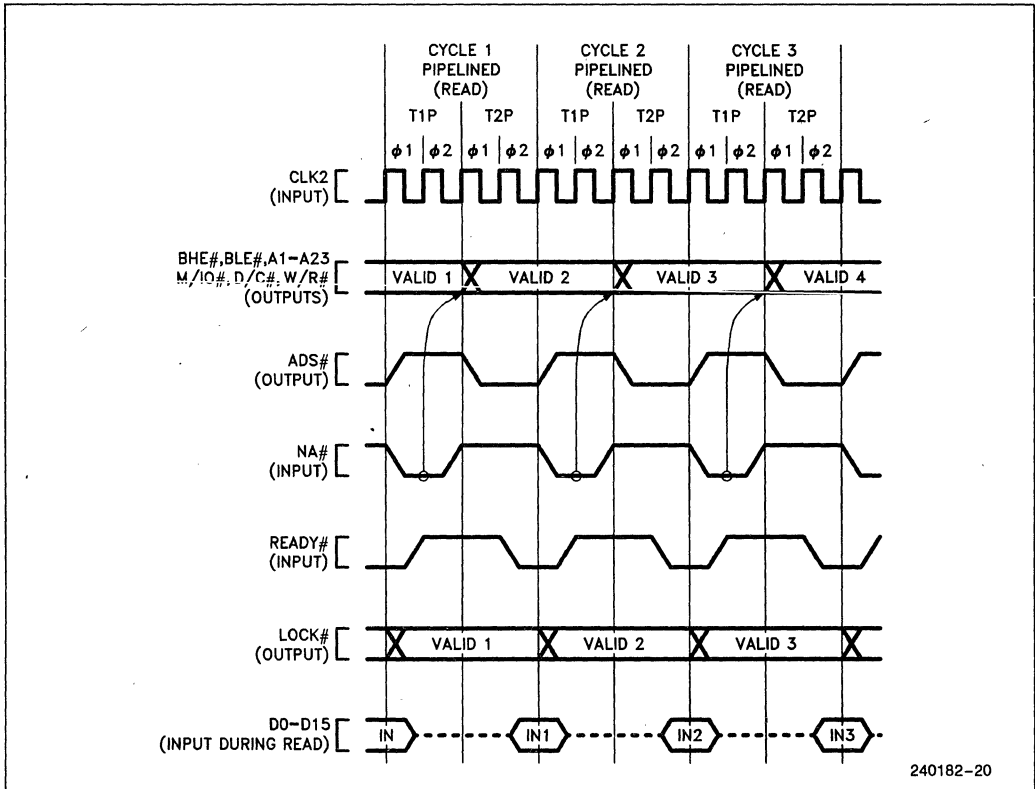


Figure 4.5. Fastest Read Cycles with Pipelined Timing

READ AND WRITE CYCLES

Data transfers occur as a result of bus cycles, classified as read or write cycles. During read cycles, data is transferred from an external device to the processor. During write cycles, data is transferred from the processor to an external device.

Two choices of bus cycle timing are dynamically selectable: non-pipelined or pipelined. After an idle bus state, the processor always uses non-pipelined timing. However the NA# (Next Address) input may be asserted to select pipelined timing for the next bus cycle. When pipelining is selected and the 80376 has a bus request pending internally, the address and definition of the next cycle is made available even before the current bus cycle is acknowledged by READY#.

Terminating a read or write cycle, like any bus cycle, requires acknowledging the cycle by asserting the READY# input. Until acknowledged, the processor inserts wait states into the bus cycle, to allow adjust-

ment for the speed of any external device. External hardware, which has decoded the address and bus cycle type, asserts the READY# input at the appropriate time.

At the end of the second bus state within the bus cycle, READY# is sampled. At that time, if external hardware acknowledges the bus cycle by asserting READY#, the bus cycle terminates as shown in Figure 4.6. If READY# is negated as in Figure 4.7, the 80376 executes another bus state (a wait state) and READY# is sampled again at the end of that state. This continues indefinitely until the cycle is acknowledged by READY# asserted.

When the current cycle is acknowledged, the 80376 terminates it. When a read cycle is acknowledged, the 80376 latches the information present at its data pins. When a write cycle is acknowledged, the write data of the 80376 remains valid throughout phase one of the next bus state, to provide write data hold time.

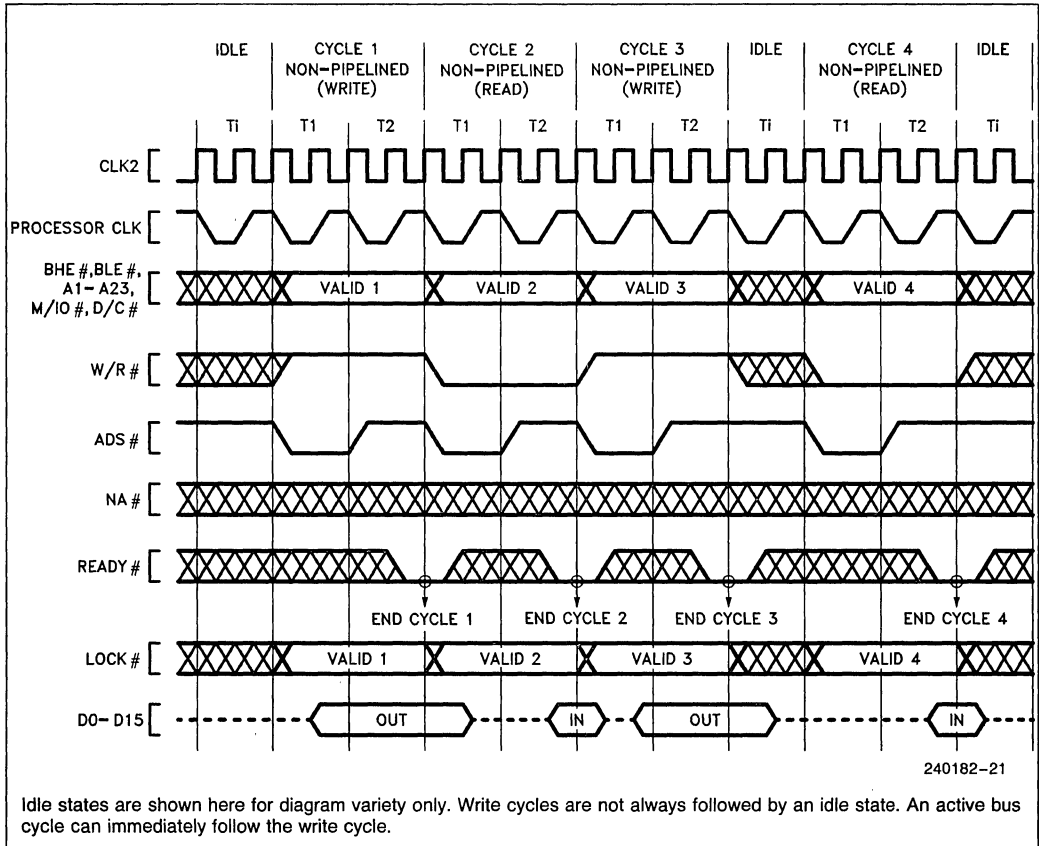


Figure 4.6. Various Non-Pipelined Bus Cycles (Zero Wait States)

Non-Pipelined Bus Cycles

Any bus cycle may be performed with non-pipelined timing. For example, Figure 4.6 shows a mixture of non-pipelined read and write cycles. Figure 4.6 shows that the fastest possible non-pipelined cycles have two bus states per bus cycle. The states are named T1 and T2. In phase one of T1, the address signals and bus cycle definition signals are driven valid and, to signal their availability, address strobe (ADS#) is simultaneously asserted.

During read or write cycles, the data bus behaves as follows. If the cycle is a read, the 80376 floats its data signals to allow driving by the external device being addressed. **The 80376 requires that all data bus pins be at a valid logic state (HIGH or LOW) at the end of each read cycle, when READY# is asserted. The system MUST be designed to meet this requirement.** If the cycle is a write, data signals are driven by the 80376 beginning in phase two of T1 until phase one of the bus state following cycle acknowledgement.

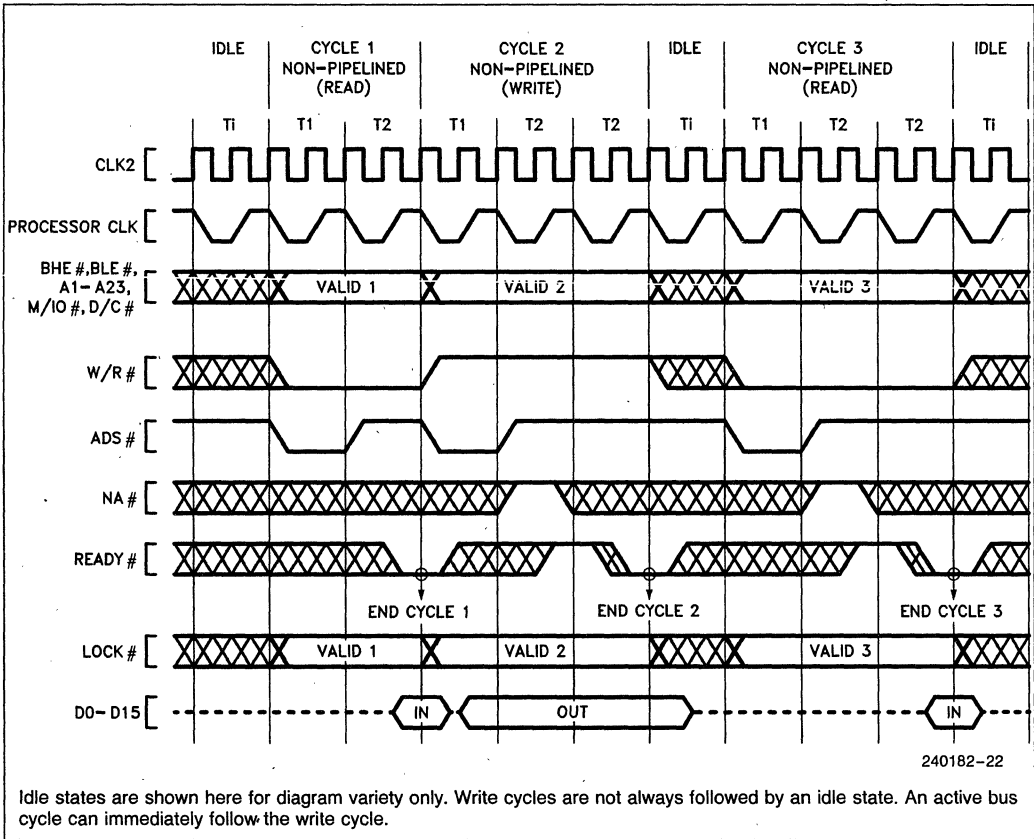


Figure 4.7. Various Non-Pipelined Bus Cycles (Various Number of Wait States)

Figure 4.7 illustrates non-pipelined bus cycles with one wait state added to Cycles 2 and 3. READY # is sampled inactive at the end of the first T2 in Cycles 2 and 3. Therefore Cycles 2 and 3 have T2 repeated again. At the end of the second T2, READY # is sampled active.

When address pipelining is not used, the address and bus cycle definition remain valid during all wait states. When wait states are added and it is desirable to maintain non-pipelined timing, it is necessary to negate NA # during each T2 state except the

last one, as shown in Figure 4.7. Cycles 2 and 3. If NA # is sampled active during a T2 other than the last one, the next state would be T2i or T2P instead of another T2.

When address pipelining is not used, the bus states and transitions are completely illustrated by Figure 4.8. The bus transitions between four possible states, T1, T2, Ti, and Th. Bus cycles consist of T1 and T2, with T2 being repeated for wait states. Otherwise the bus may be idle, Ti, or in the hold acknowledge state Th.

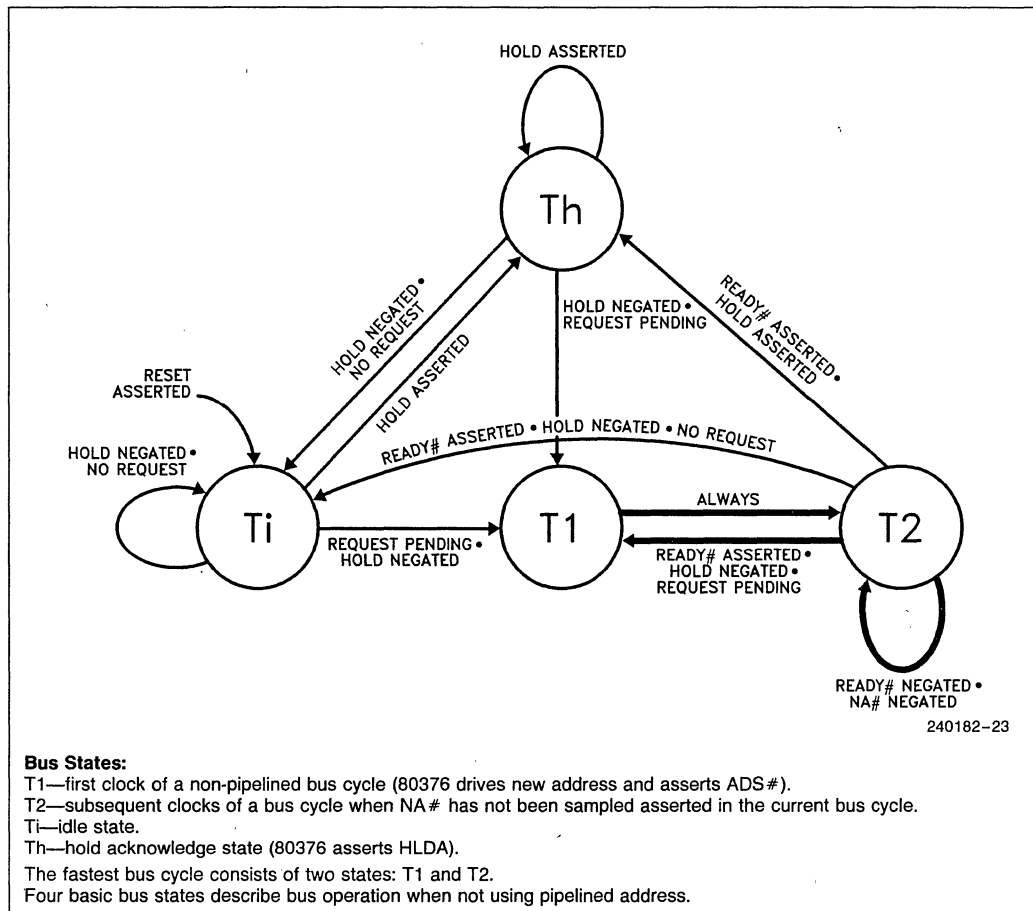


Figure 4.8. 80376 Bus States (Not Using Pipelined Address)

Bus cycles always begin with T1. T1 always leads to T2. If a bus cycle is not acknowledged during T2 and NA# is inactive, T2 is repeated. When a cycle is acknowledged during T2, the following state will be T1 of the next bus cycle if a bus request is pending internally, or Ti if there is no bus request pending, or Th if the HOLD input is being asserted.

Use of pipelining allows the 80376 to enter three additional bus states not shown in Figure 4.8. Figure 4.12 on page 49 is the complete bus state diagram, including pipelined cycles.

Pipelined Bus Cycles

Pipelining is the option of requesting the address and the bus cycle definition of the next inter-

nally pending bus cycle before the current bus cycle is acknowledged with READY# asserted. ADS# is asserted by the 80376 when the next address is issued. The pipelining option is controlled on a cycle-by-cycle basis with the NA# input signal.

Once a bus cycle is in progress and the current address has been valid for at least one entire bus state, the NA# input is sampled at the end of every phase one until the bus cycle is acknowledged. During non-pipelined bus cycles NA# is sampled at the end of phase one in every T2. An example is Cycle 2 in Figure 4.9, during which NA# is sampled at the end of phase one of every T2 (it was asserted once during the first T2 and has no further effect during that bus cycle).

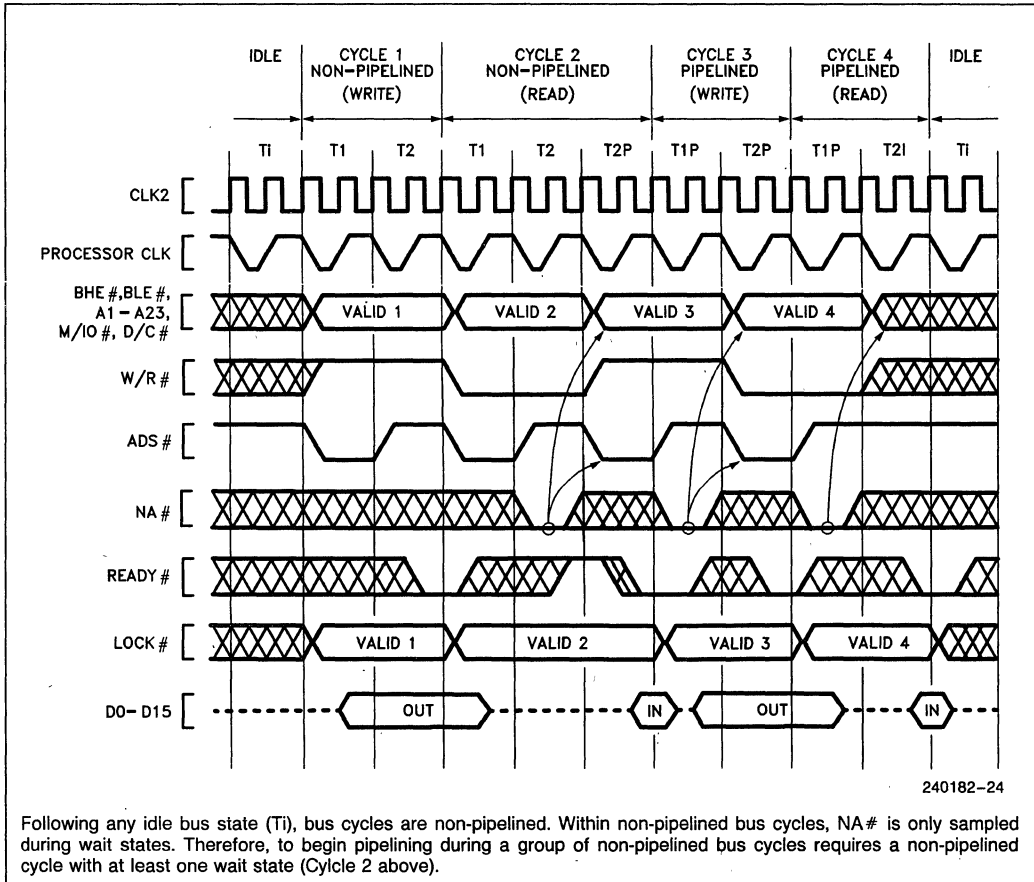


Figure 4.9. Transitioning to Pipelining during Burst of Bus Cycles

If NA# is sampled active, the 80376 is free to drive the address and bus cycle definition of the next bus cycle, and assert ADS#, as soon as it has a bus request internally pending. It may drive the next address as early as the next bus state, whether the current bus cycle is acknowledged at that time or not.

Regarding the details of pipelining, the 80376 has the following characteristics:

1. The next address and status may appear as early as the bus state after NA# was sampled active (see Figures 4.9 or 4.10). In that case, state T2P is entered immediately. However, when there is not an internal bus request already pending, the next address and status will not be available immediately after NA# is asserted and T2I is entered instead of T2P (see Figure 4.11 Cycle 3). Provided the current bus cycle isn't yet acknow-

ledged by READY# asserted, T2P will be entered as soon as the 80376 does drive the next address and status. External hardware should therefore observe the ADS# output as confirmation the next address and status are actually being driven on the bus.

2. Any address and status which are validated by a pulse on the 80376 ADS# output will remain stable on the address pins for at least two processor clock periods. The 80376 cannot produce a new address and status more frequently than every two processor clock periods (see Figures 4.9, 4.10 and 4.11).
3. Only the address and bus cycle definition of the very next bus cycle is available. The pipelining capability cannot look further than one bus cycle ahead (see Figure 4.11, Cycle 1).

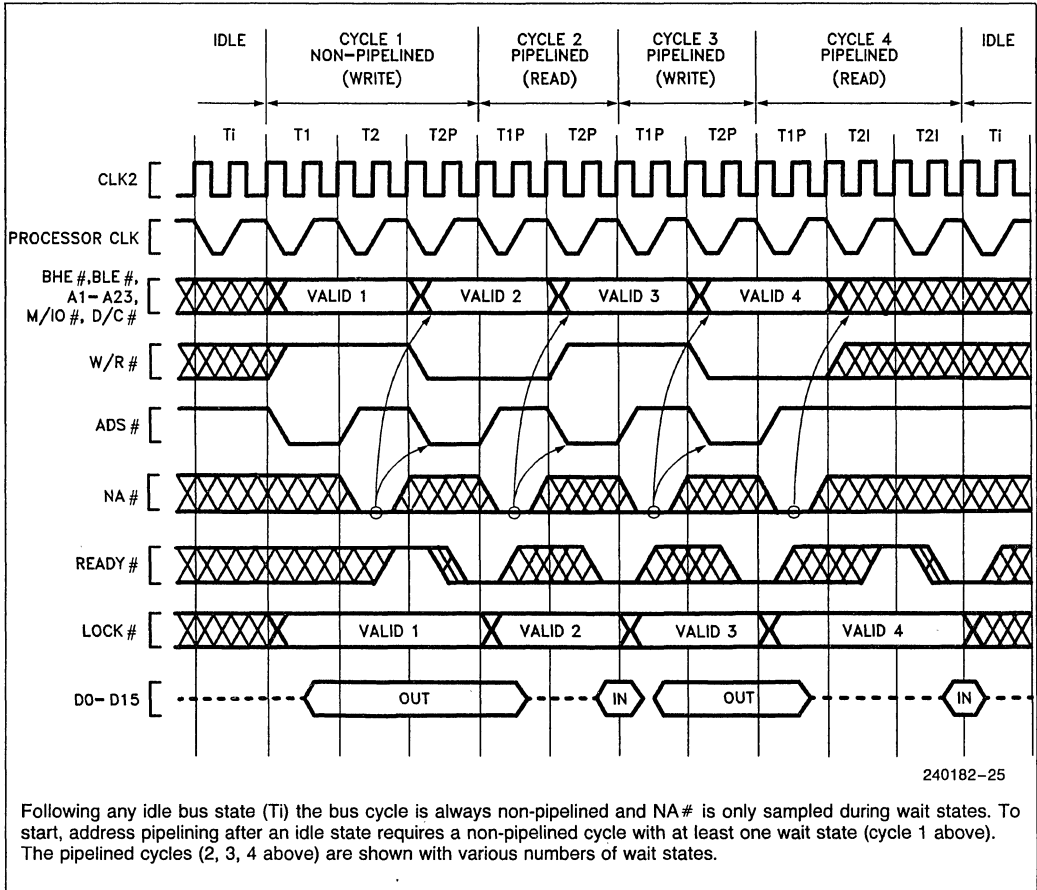


Figure 4.10. Fastest Transition to Pipelined Bus Cycle Following Idle Bus State

The complete bus state transition diagram, including pipelining is given by Figure 4.12. Note it is a superset of the diagram for non-pipelined only, and the three additional bus states for pipelining are drawn in bold.

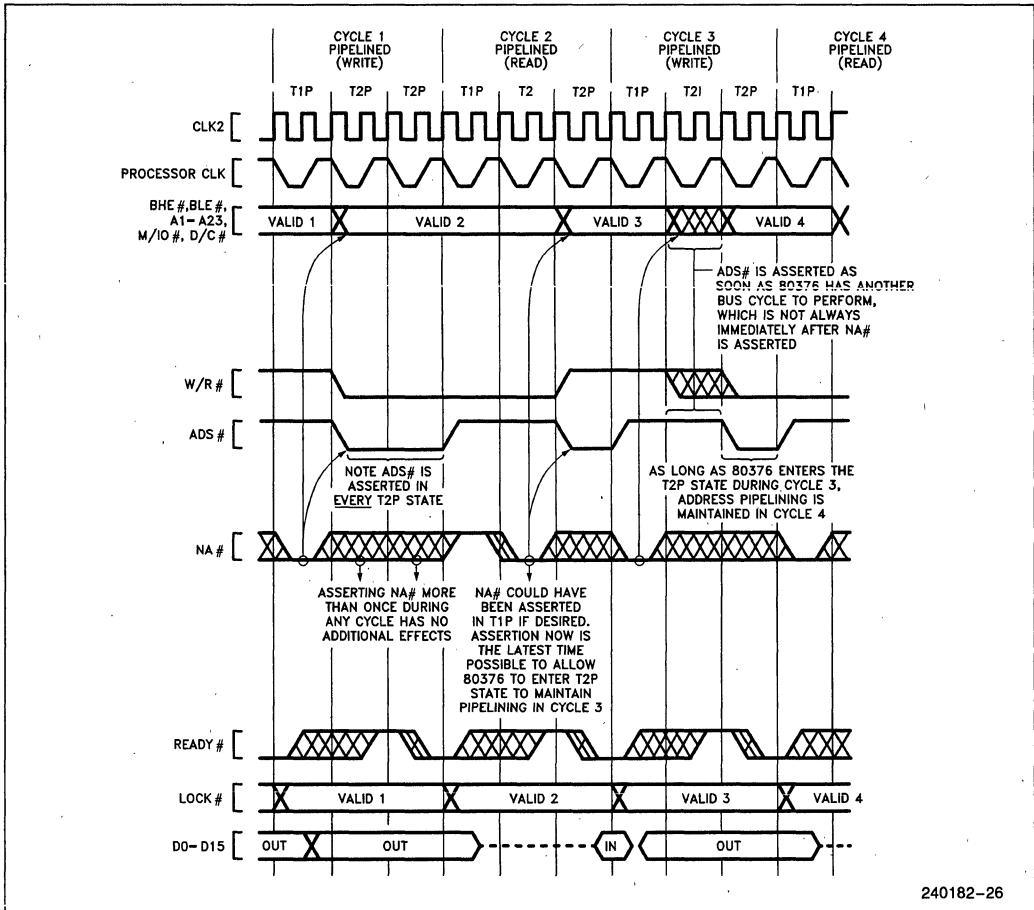
The fastest bus cycle with pipelining consists of just two bus states, T1P and T2P (recall for non-pipelined it is T1 and T2). T1P is the first bus state of a pipelined cycle.

Initiating and Maintaining Pipelined Bus Cycles

Using the state diagram Figure 4.12, observe the transitions from an idle state, Ti, to the beginning of

a pipelined bus cycle T1P. From an idle state, Ti, the first bus cycle must begin with T1, and is therefore a non-pipelined bus cycle. The next bus cycle will be pipelined, however, provided NA# is asserted and the first bus cycle ends in a T2P state (the address and status for the next bus cycle is driven during T2P). The fastest path from an idle state to a pipelined bus cycle is shown in bold below:

Ti, Ti,	T1-T2-T2P,	T1P-T2P,
idle states	non-pipelined cycle	pipelined cycle



240182-26

Figure 4.11. Details of Address Pipelining during Cycles with Wait States

T1-T2-T2P are the states of the bus cycle that establishes address pipelining for the next bus cycle, which begins with T1P. The same is true after a bus hold state, shown below:

T_h, T_h, T_h T1-T2-T2P, T1P-T2P,

hold acknowledge non-pipelined cycle pipelined cycle

The transition to pipelined address is shown functionally by Figure 4.10, Cycle 1. Note that Cycle 1 is used to transition into pipelined address timing for the subsequent Cycles 2, 3 and 4, which are pipelined. The NA# input is asserted at the appropriate time to select address pipelining for Cycles 2, 3 and 4.

Once a bus cycle is in progress and the current address and status has been valid for one entire bus cycle, the NA# input is sampled at the end of every phase one until the bus cycle is acknowledged.

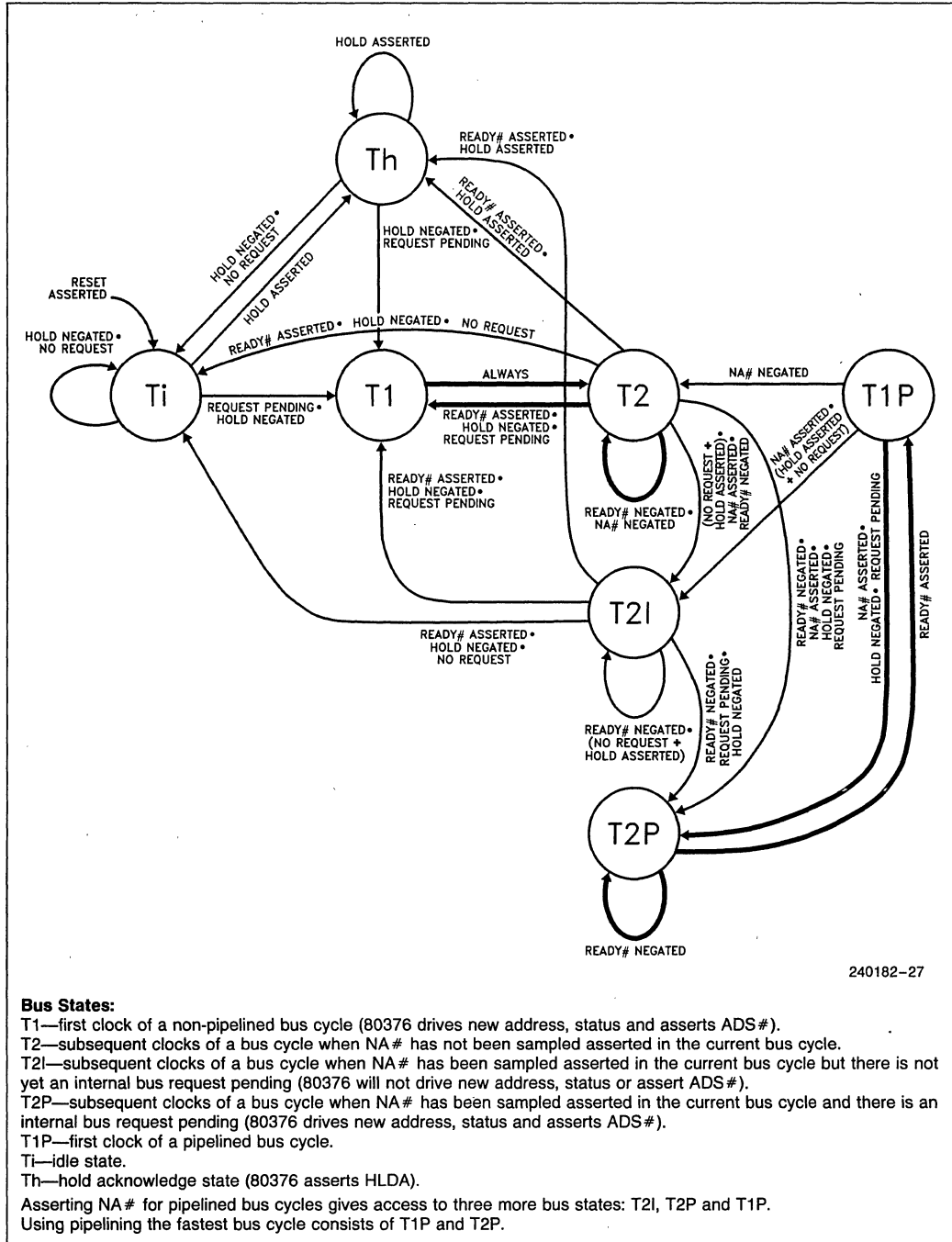


Figure 4.12. 80376 Processor Complete Bus States (Including Pipelining)

Sampling begins in T2 during Cycle 1 in Figure 4.10. Once NA# is sampled active during the current cycle, the 80376 is free to drive a new address and bus cycle definition on the bus as early as the next bus state. In Figure 4.10, Cycle 1 for example, the next address and status is driven during state T2P. Thus Cycle 1 makes the transition to pipelined timing, since it begins with T1 but ends with T2P. Because the address for Cycle 2 is available before Cycle 2 begins, Cycle 2 is called a pipelined bus cycle, and it begins with T1P. Cycle 2 begins as soon as READY# asserted terminates Cycle 1.

Examples of transition bus cycles are Figure 4.10, Cycle 1 and Figure 4.9, Cycle 2. Figure 4.10 shows transition during the very first cycle after an idle bus state, which is the fastest possible transition into address pipelining. Figure 4.9, Cycle 2 shows a transition cycle occurring during a burst of bus cycles. In any case, a transition cycle is the same whenever it occurs: it consists at least of T1, T2 (NA# is asserted at that time), and T2P (provided the 80376 has an internal bus request already pending, which it almost always has). T2P states are repeated if wait states are added to the cycle.

Note that only three states (T1, T2 and T2P) are required in a bus cycle performing a **transition** from non-pipelined into pipelined timing, for example Figure 4.10, Cycle 1. Figure 4.10, Cycles 2, 3 and 4 show that pipelining can be maintained with two-state bus cycles consisting only of T1P and T2P.

Once a pipelined bus cycle is in progress, pipelined timing is maintained for the next cycle by asserting NA# and detecting that the 80376 enters T2P during the current bus cycle. The current bus cycle must end in state T2P for pipelining to be maintained in the next cycle. T2P is identified by the assertion of ADS#. Figures 4.9 and 4.10 however, each show

pipelining ending after Cycle 4 because Cycle 4 ends in T2I. This indicates the 80376 didn't have an internal bus request prior to the acknowledgement of Cycle 4. If a cycle ends with a T2 or T2I, the next cycle will not be pipelined.

Realistically, pipelining is almost always maintained as long as NA# is sampled asserted. This is so because in the absence of any other request, a code prefetch request is always internally pending until the instruction decoder and code prefetch queue are completely full. Therefore pipelining is maintained for long bursts of bus cycles, if the bus is available (i.e., HOLD inactive) and NA# is sampled active in each of the bus cycles.

INTERRUPT ACKNOWLEDGE (INTA) CYCLES

In response to an interrupt request on the INTR input when interrupts are enabled, the 80376 performs two interrupt acknowledge cycles. These bus cycles are similar to read cycles in that bus definition signals define the type of bus activity taking place, and each cycle continues until acknowledged by READY# sampled active.

The state of A₂ distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A₂₃-A₃, A₁, BLE# LOW, A₂ and BHE# HIGH). The byte address driven during the second interrupt acknowledge cycle is 0 (A₂₃-A₁, BLE# LOW and BHE# HIGH).

The LOCK# output is asserted from the beginning of the first interrupt acknowledge cycle until the end of the second interrupt acknowledge cycle. Four idle bus states, T_i, are inserted by the 80376 between the two interrupt acknowledge cycles for compatibility with the interrupt specification T_{RHRL} of the 8259A Interrupt Controller and the 82370 Integrated Peripheral.

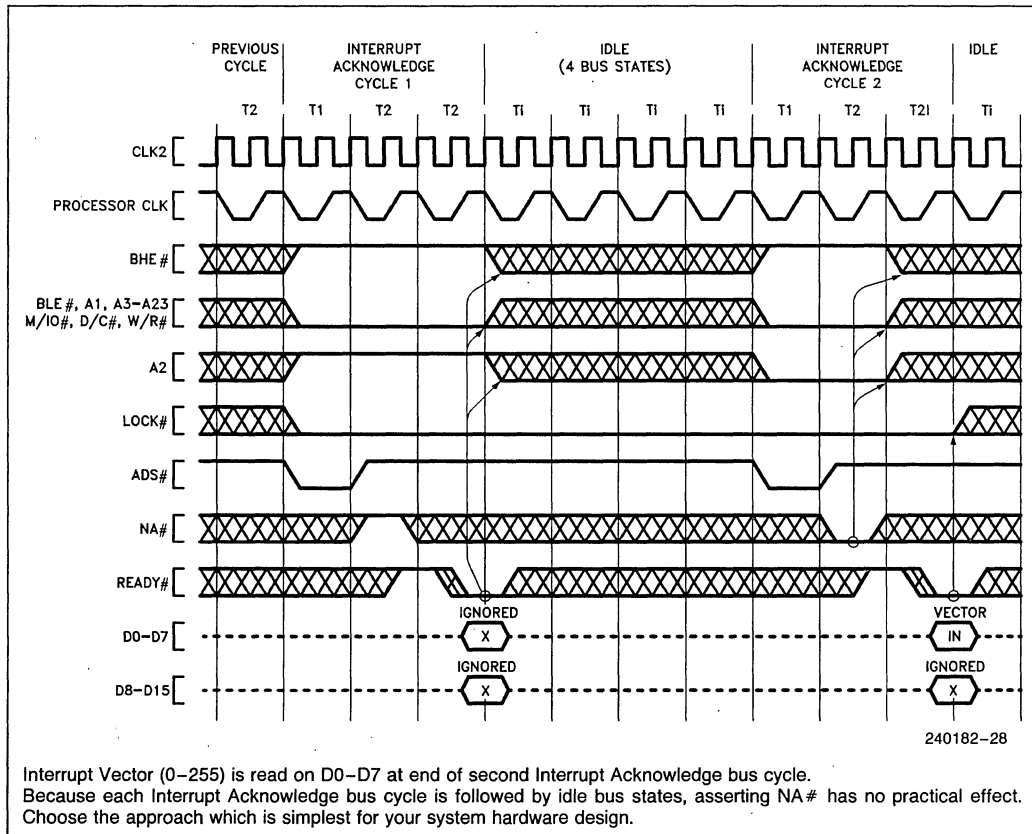
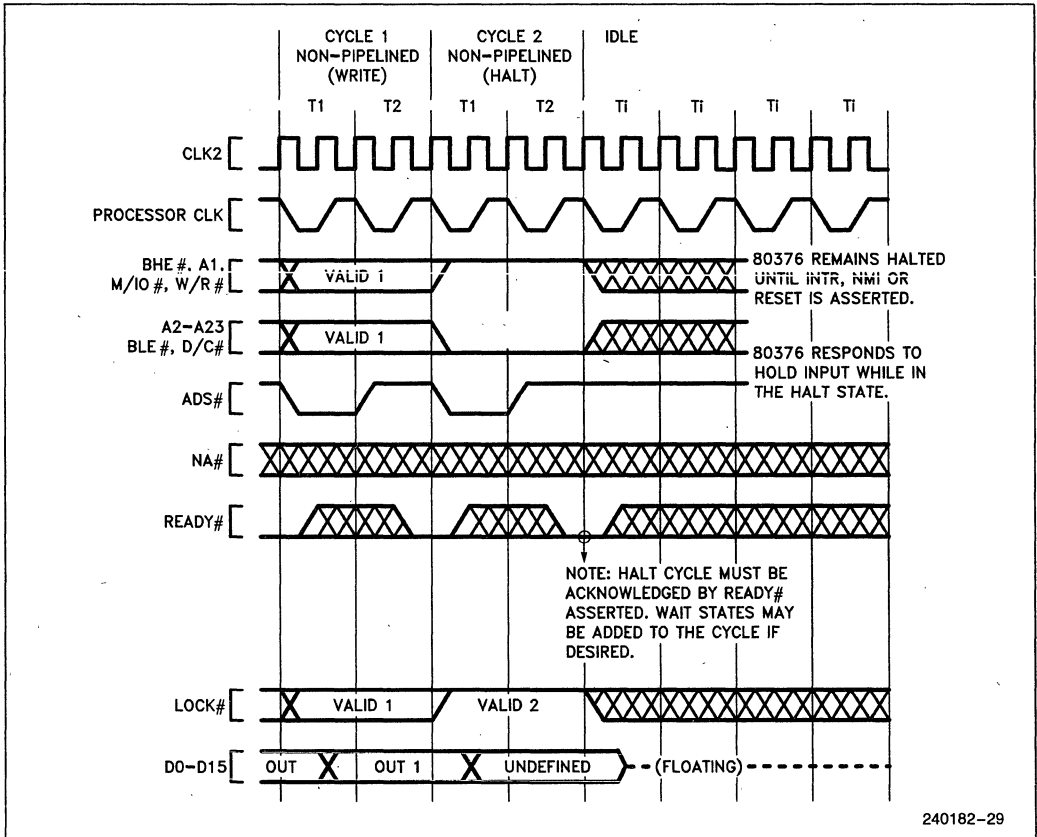


Figure 4.13. Interrupt Acknowledge Cycles

During both interrupt acknowledge cycles, D₁₅-D₀ float. No data is read at the end of the first interrupt acknowledge cycle. At the end of the second interrupt acknowledge cycle, the 80376 will read an external interrupt vector from D₇-D₀ of the data bus. The vector indicates the specific interrupt number (from 0-255) requiring service.

HALT INDICATION CYCLE

The 80376 execution unit halts as a result of executing a HLT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the state of the bus definition signals shown on page 34, **Bus Cycle Definition Signals**, and a byte address of 2. The halt indication cycle must be acknowledged by READY# asserted. A halted 80376 resumes execution when INTR (if interrupts are enabled), NMI or RESET is asserted.



240182-29

Figure 4.14. Example Halt Indication Cycle from Non-Pipelined Cycle

SHUTDOWN INDICATION CYCLE

The 80376 shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the state of the bus definition signals shown on page 34 **Bus Cycle Definition Signals** and a byte address of 0. The shutdown indication cycle must be acknowledged by READY# asserted. A shutdown 80376 resumes execution when NMI or RESET is asserted.

ENTERING AND EXITING HOLD ACKNOWLEDGE

The bus hold acknowledge state, T_h , is entered in response to the HOLD input being asserted. In the bus hold acknowledge state, the 80376 floats all outputs or bidirectional signals, except for HLDA. HLDA is asserted as long as the 80376 remains in the bus hold acknowledge state. In the bus hold acknowledge state, all inputs except HOLD and RESET are ignored.

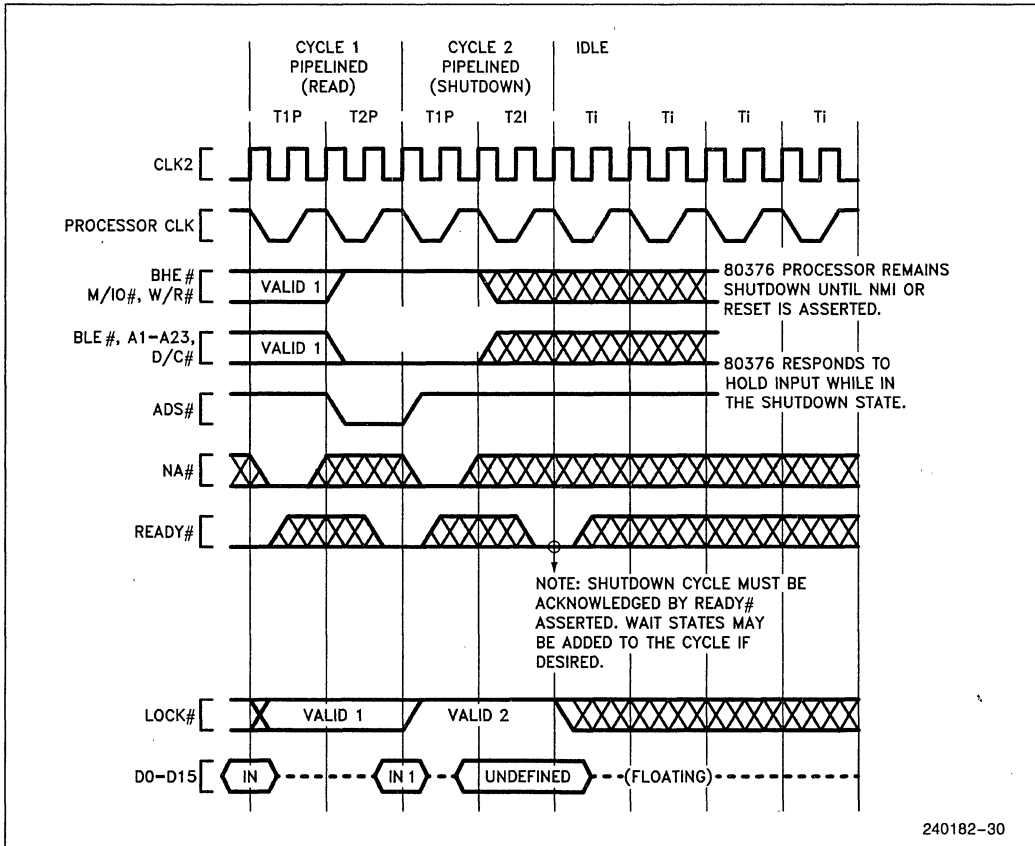


Figure 4.15. Example Shutdown Indication Cycle from Non-Pipelined Cycle

T_h may be entered from a bus idle state as in Figure 4.16 or after the acknowledgement of the current physical bus cycle if the LOCK# signal is not asserted, as in Figures 4.17 and 4.18.

T_h is exited in response to the HOLD input being negated. The following state will be T_i as in Figure 4.16 if no bus request is pending. The following bus

state will be T_1 if a bus request is internally pending, as in Figures 4.17 and 4.18. T_h is exited in response to RESET being asserted.

If a rising edge occurs on the edge-triggered NMI input while in T_h , the event is remembered as a non-maskable interrupt 2 and is serviced when T_h is exited unless the 80376 is reset before T_h is exited.

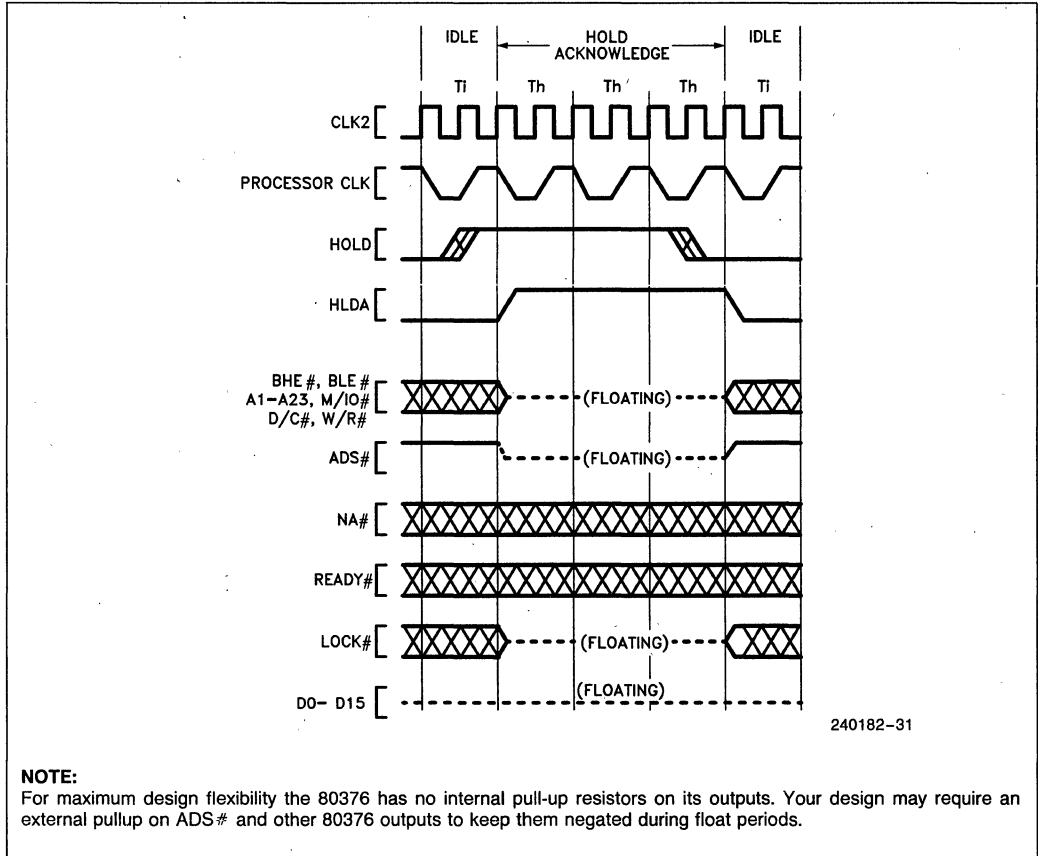


Figure 4.16. Requesting Hold from Idle Bus

RESET DURING HOLD ACKNOWLEDGE

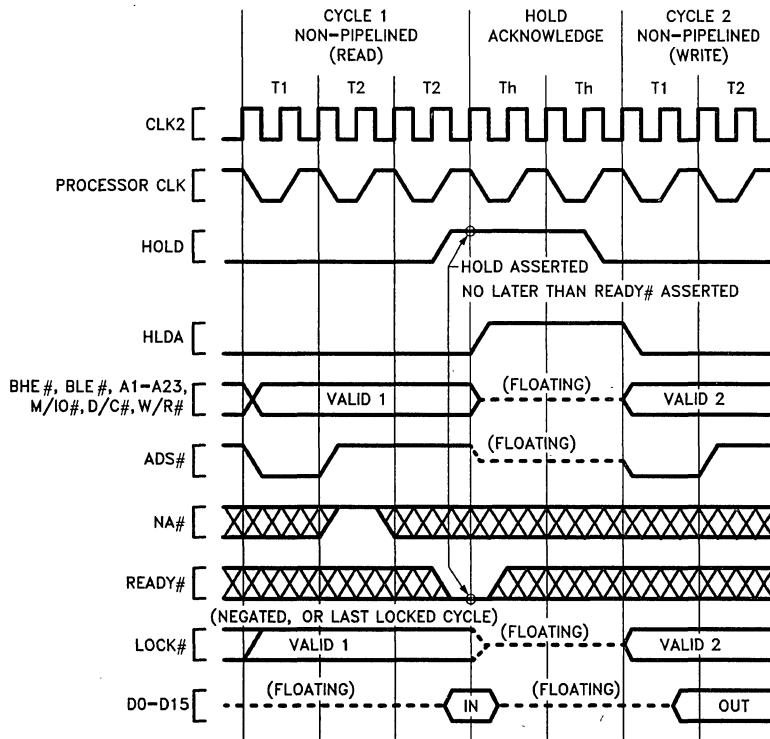
RESET being asserted takes priority over HOLD being asserted. If RESET is asserted while HOLD remains asserted, the 80376 drives its pins to defined states during reset, as in Table 4.5, Pin State During Reset, and performs internal reset activity as usual.

If HOLD remains asserted when RESET is inactive, the 80376 enters the hold acknowledge state before performing its first bus cycle, provided HOLD is still asserted when the 80376 processor would other-

wise perform its first bus cycle. If HOLD remains asserted when RESET is inactive, the BUSY# input is still sampled as usual to determine whether a self test is being requested.

BUS ACTIVITY DURING AND FOLLOWING RESET

RESET is the highest priority input signal, capable of interrupting any processor activity when it is asserted. A bus cycle in progress can be aborted at any stage, or idle states or bus hold acknowledge states discontinued so that the reset state is established.



240182-32

NOTE:

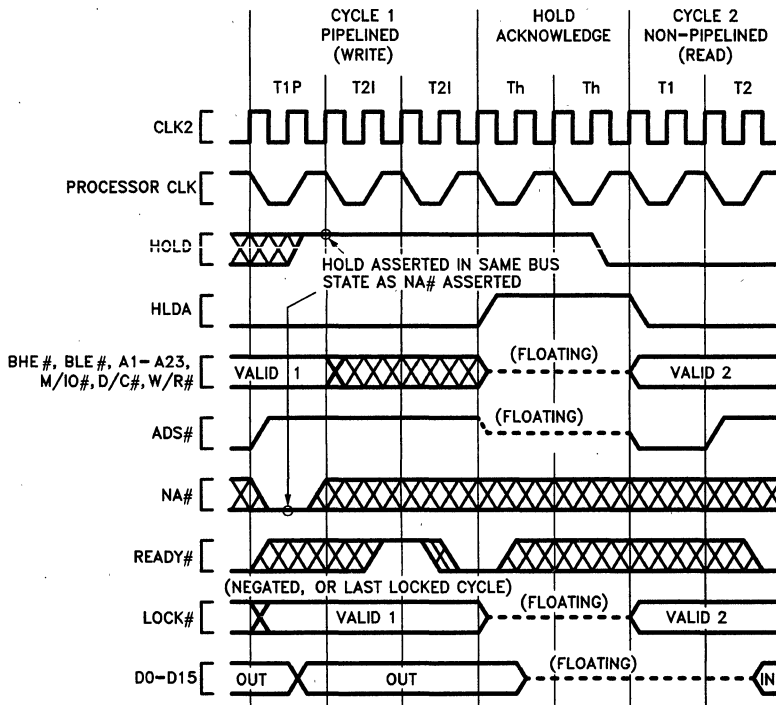
HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t_{23} and t_{24}) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

Figure 4.17. Requesting Hold from Active Bus (NA # Inactive)

RESET should remain asserted for at least 15 CLK2 periods to ensure it is recognized throughout the 80376, and at least 80 CLK2 periods if a 80376 self-test is going to be requested at the falling edge. RESET asserted pulses less than 15 CLK2 periods may not be recognized. RESET pulses less than 80 CLK2

periods followed by a self-test may cause the self-test to report a failure when no true failure exists.

Provided the RESET falling edge meets setup and hold times t_{25} and t_{26} , the internal processor clock phase is defined at that time as illustrated by Figure 4.19 and Figure 6.7.



240182-33

NOTE:

HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t_{23} and t_{24}) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

Figure 4.18. Requesting Hold from Idle Bus (NA # Active)

An 80376 self-test may be requested at the time RESET goes inactive by having the BUSY# input at a LOW level as shown in Figure 4.19. The self-test requires $(2^{20} + \text{approximately } 60)$ CLK2 periods to complete. The self-test duration is not affected by the test results. Even if the self-test indicates a

problem, the 80376 attempts to proceed with the reset sequence afterwards.

After the RESET falling edge (and after the self-test if it was requested) the 80376 performs an internal initialization sequence for approximately 350 to 450 CLK2 periods.

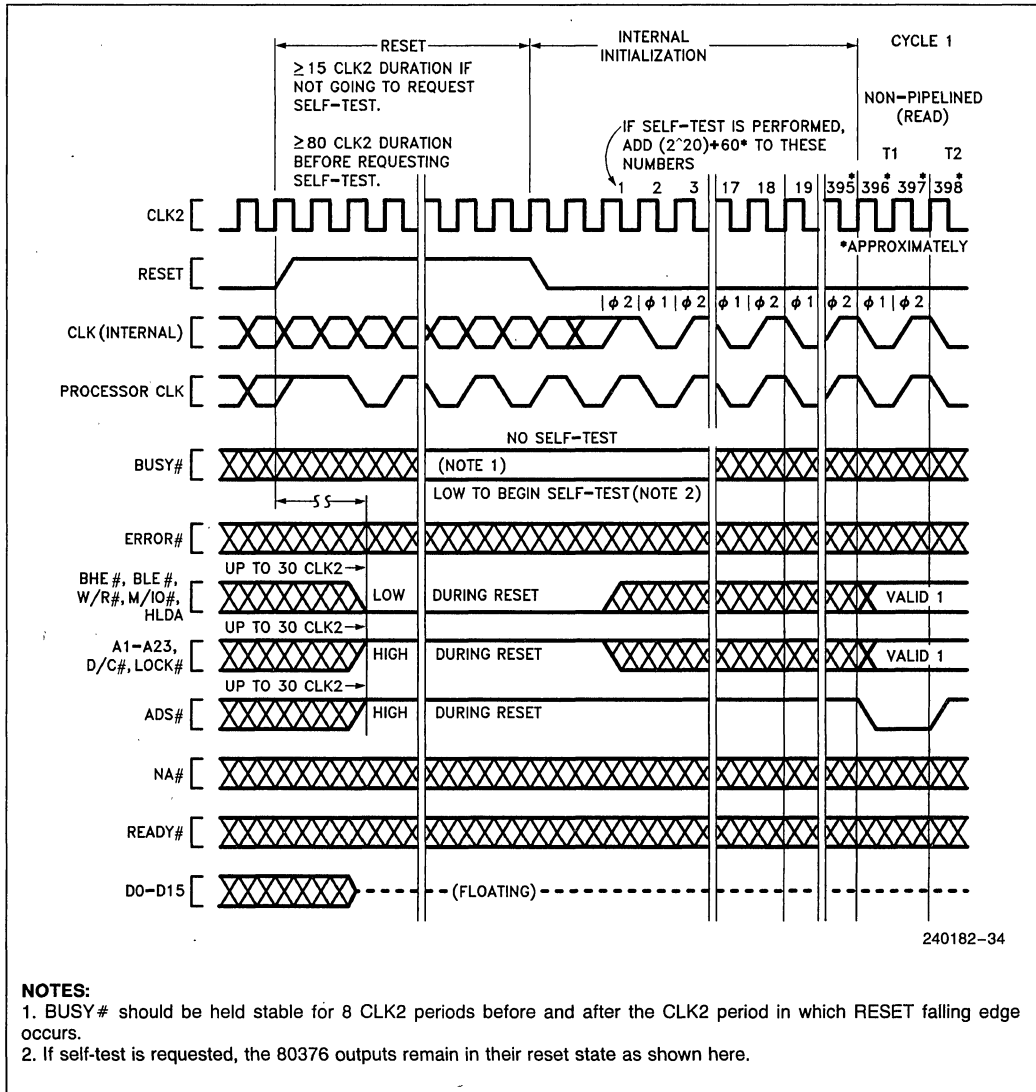


Figure 4.19. Bus Activity from Reset until First Code Fetch

4.5 Self-Test Signature

Upon completion of self-test (if self-test was requested by driving BUSY# LOW at the falling edge of RESET) the EAX register will contain a signature of 00000000H indicating the 80376 passed its self-test of microcode and major PLA contents with no problems detected. The passing signature in EAX, 00000000H, applies to all 80376 revision levels. Any non-zero signature indicates the 80376 unit is faulty.

4.6 Component and Revision Identifiers

To assist 80376 users, the 80376 after reset holds a component identifier and revision identifier in its DX register. The upper 8 bits of DX hold 33H as identification of the 80376 component. (The lower nibble, 03H, refers to the Intel386™ architecture. The upper nibble, 30H, refers to the third member of the Intel386 family). The lower 8 bits of DX hold an 8-bit unsigned binary number related to the

component revision level. The revision identifier will, in general, chronologically track those component steppings which are intended to have certain improvements or distinction from previous steppings. The 80376 revision identifier will track that of the 80386 where possible.

The revision identifier is intended to assist 80376 users to a practical extent. However, the revision identifier value is not guaranteed to change with every stepping revision, or to follow a completely uniform numerical sequence, depending on the type or intention of revision, or manufacturing materials required to be changed. Intel has sole discretion over these characteristics of the component.

Table 4.7. Component and Revision Identifier History

80376 Stepping Name	Revision Identifier
A0	05H

4.7 Coprocessor Interfacing

The 80376 provides an automatic interface for the Intel 80387SX numeric floating-point coprocessor. The 80387SX coprocessor uses an I/O mapped interface driven automatically by the 80376 and assisted by three dedicated signals: BUSY#, ERROR# and PEREQ.

As the 80376 begins supporting a coprocessor instruction, it tests the BUSY# and ERROR# signals to determine if the coprocessor can accept its next instruction. Thus, the BUSY# and ERROR# inputs eliminate the need for any "preamble" bus cycles for communication between processor and coprocessor. The 80387SX can be given its command opcode immediately. The dedicated signals provide instruction synchronization, and eliminate the need of using the 80376 WAIT opcode (9BH) for 80387SX instruction synchronization (the WAIT opcode was required when the 8086 or 8088 was used with the 8087 coprocessor).

Custom coprocessors can be included in 80376 based systems by memory-mapped or I/O-mapped interfaces. Such coprocessor interfaces allow a completely custom protocol, and are not limited to a set of coprocessor protocol "primitives". Instead, memory-mapped or I/O-mapped interfaces may use all applicable 80376 instructions for high-speed coprocessor communication. The BUSY# and

ERROR# inputs of the 80376 may also be used for the custom coprocessor interface, if such hardware assist is desired. These signals can be tested by the 80376 WAIT opcode (9BH). The WAIT instruction will wait until the BUSY# input is inactive (interruptable by an NMI or enabled INTR input), but generates an exception 16 fault if the ERROR# pin is active when the BUSY# goes (or is) inactive. If the custom coprocessor interface is memory-mapped, protection of the addresses used for the interface can be provided with the segmentation mechanism of the 80376. If the custom interface is I/O-mapped, protection of the interface can be provided with the 80376 IOPL (I/O Privilege Level) mechanism.

The 80387SX numeric coprocessor interface is I/O mapped as shown in Table 4.8. Note that the 80387SX coprocessor interface addresses are beyond the 0H-0FFFFH range for programmed I/O. When the 80376 supports the 80387SX coprocessor, the 80376 automatically generates bus cycles to the coprocessor interface addresses.

Table 4.8 Numeric Coprocessor Port Addresses

Address in 80376 I/O Space	80387SX Coprocessor Register
8000F8H	Opcode Register
8000FCH	Operand Register
8000FEH	Operand Register

SOFTWARE TESTING FOR COPROCESSOR PRESENCE

When software is used to test coprocessor (80387SX) presence, it should use only the following coprocessor opcodes: FNINIT, FNSTCW and FNSTSW. To use other coprocessor opcodes when a coprocessor is known to be not present, first set EM = 1 in the 80376 CR0 register.

5.0 PACKAGE THERMAL SPECIFICATIONS

The Intel 80376 embedded processor is specified for operation when case temperature is within the range of 0°C–115°C for the ceramic 88-pin PGA package, and 0°C–110°C for the 100-pin plastic package. The case temperature may be measured in any environment, to determine whether the 80376 is within specified operating range. The case temperature should be measured at the center of the top surface.

The ambient temperature is guaranteed as long as T_c is not violated. The ambient temperature can be calculated from the θ_{jc} and θ_{ja} from the following equations:

$$T_J = T_c + P \cdot \theta_{jc}$$

$$T_A = T_J - P \cdot \theta_{ja}$$

$$T_c = T_A + P \cdot [\theta_{ja} - \theta_{jc}]$$

Values for θ_{ja} and θ_{jc} are given in Table 5.1 for the 100-lead fine pitch. θ_{ja} is given at various airflows. Table 5.2 shows the maximum T_a allowable (without exceeding T_c) at various airflows. Note that T_a can be improved further by attaching "fins" or a "heat sink" to the package. P is calculated using the maximum *hot* I_{CC} .

Table 5.1. 80376 Package Thermal Characteristics Thermal Resistances ($^{\circ}\text{C}/\text{Watt}$) θ_{jc} and θ_{ja}

Package	θ_{jc}	θ_{ja} Versus Airflow-ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100-Lead Fine Pitch	7	33	27	24	21	18	17
88-Pin PGA	2	25	20	17	14	12	11

Assuming I_{CC} hot of 360 mA, V_{CC} of 5.0V, and a T_{CASE} of 110°C for plastic and 115°C for the 88-Pin PGA Package:

Table 5.2. 80376 Maximum Allowable Ambient Temperature at Various Airflows

Package	θ_{jc}	$T_A(^{\circ}\text{C})$ vs Airflow-ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100-Lead Fine Pitch	7	63	74	79	85	91	92
88-Pin PGA	2	74	83	88	93	97	99

6.0 ELECTRICAL SPECIFICATIONS

The following sections describe recommended electrical connections for the 80376, and its electrical specifications.

6.1 Power and Grounding

The 80376 is implemented in CHMOS III technology and has modest power requirements. However, its high clock frequency and 47 output buffers (address, data, control, and HLDA) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 14 V_{CC} and 18 V_{SS} pins separately feed functional units of the 80376.

Power and ground connections must be made to all external V_{CC} and GND pins of the 80376. On the circuit board, all V_{CC} pins should be connected on a V_{CC} plane and all V_{SS} pins should be connected on a GND plane.

POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitors should be placed near the 80376. The 80376 driving its 24-bit address bus and 16-bit data bus at high frequencies can cause transient power surges, particularly when driving large capacitive loads. Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the 80376 and decoupling capacitors as much as possible.

RESISTOR RECOMMENDATIONS

The ERROR# and BUSY# inputs have internal pull-up resistors of approximately $20\text{ K}\Omega$ and the PEREQ input has an internal pull-down resistor of approximately $20\text{ K}\Omega$ built into the 80376 to keep these signals inactive when the 80387SX is not present in the system (or temporarily removed from its socket).

In typical designs, the external pull-up resistors shown in Table 6.1 are recommended. However, a particular design may have reason to adjust the resistor values recommended here, or alter the use of pull-up resistors in other ways.

Table 6.1. Recommended Resistor Pull-Ups to V_{CC}

Pin	Signal	Pull-Up Value	Purpose
16	ADS#	20 K Ω \pm 10%	Lightly Pull ADS# Inactive during 80376 Hold Acknowledge States
26	LOCK#	20 K Ω \pm 10%	Lightly Pull LOCK# Inactive during 80376 Hold Acknowledge States

OTHER CONNECTION RECOMMENDATIONS

For reliable operation, always connect unused inputs to an appropriate signal level. N/C pins should always remain **unconnected**. **Connection of N/C pins to V_{CC} or V_{SS} will result in incompatibility with future steppings of the 80376.**

Particularly when not using interrupts or bus hold (as when first prototyping), prevent any chance of spurious activity by connecting these associated inputs to GND:

- INTR
- NMI
- HOLD

If not using address pipelining connect the NA# pin to a pull-up resistor in the range of 20 K Ω to V_{CC}.

6.2 Absolute Maximum Ratings

Table 6.2. Maximum Ratings

Parameter	Maximum Rating
Storage Temperature	−65°C to +150°C
Case Temperature under Bias	−65°C to +120°C
Supply Voltage with Respect to V _{SS}	−0.5V to +6.5V
Voltage on Other Pins	−0.5V to (V _{CC} + 0.5)V

Table 6.2 gives a stress ratings only, and functional operation at the maximums is not guaranteed. Functional operating conditions are given in **Section 6.3, D.C. Specifications**, and **Section 6.4, A.C. Specifications**.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the 80376 contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

6.3 D.C. Specifications

ADVANCE INFORMATION SUBJECT TO CHANGE
Table 6.3: 80376 D.C. Characteristics

 Functional Operating Range: $V_{CC} = 5V \pm 10\%$; $T_{CASE} = 0^{\circ}C$ to $115^{\circ}C$ 88-pin PGA, $T_{CASE} = 0^{\circ}C$ to $110^{\circ}C$ 100-pin plastic

Symbol	Parameter	Min	Max	Unit
V_{IL}	Input LOW Voltage	-0.3	+0.8	V(1)
V_{IH}	Input HIGH Voltage	2.0	$V_{CC} + 0.3$	V(1)*
V_{ILC}	CLK2 Input LOW Voltage	-0.3	+0.8	V(1)
V_{IHC}	CLK2 Input HIGH Voltage	$V_{CC} - 0.8$	$V_{CC} + 0.3$	V(1)
V_{OL}	Output LOW Voltage			
$I_{OL} = 4 \text{ mA}$:	A ₂₃ -A ₁ , D ₁₅ -D ₀		0.45	V(1)
$I_{OL} = 5 \text{ mA}$:	BHE#, BLE#, W/R#, D/C#, M/IO#, LOCK#, ADS#, HLDA		0.45	V(1)
V_{OH}	Output High Voltage			
$I_{OH} = -1 \text{ mA}$:	A ₂₃ -A ₁ , D ₁₅ -D ₀	2.4		V(1)
$I_{OH} = -0.2 \text{ mA}$:	A ₂₃ -A ₁ , D ₁₅ -D ₀	$V_{CC} - 0.5$		V(1)
$I_{OH} = -0.9 \text{ mA}$:	BHE#, BLE#, W/R#, D/C#, M/IO#, LOCK#, ADS#, HLDA	2.4		V(1)
$I_{OH} = -0.18 \text{ mA}$:	BHE#, BLE#, W/R#, D/C#, M/IO#, LOCK#, ADS#, HLDA	$V_{CC} - 0.5$		V(1)
I_{LI}	* Input Leakage Current (For All Pins except PEREQ, BUSY# and ERROR#)		± 15	μA, $0V \leq V_{IN} \leq V_{CC}^{(1)}$
I_{IH}	Input Leakage Current (PEREQ Pin)		200	μA, $V_{IH} = 2.4V^{(1, 2)}$
I_{IL}	Input Leakage Current (Busy# and ERROR# Pins)		-400	μA, $V_{IL} = 0.45V^{(3)}$
I_{LO}	Output Leakage Current		± 15	μA, $0.45V \leq V_{OUT} \leq V_{CC}^{(1)}$
I_{CC}	Supply Current at HOT		400 360	mA ⁽⁴⁾ mA ⁽⁶⁾
C_{IN}	Input Capacitance		10	pF, $F_C = 1 \text{ MHz}^{(5)}$
C_{OUT}	Output or I/O Capacitance		12	pF, $F_C = 1 \text{ MHz}^{(5)}$
C_{CLK}	CLK2 Capacitance		20	pF, $F_C = 1 \text{ MHz}^{(5)}$

NOTES:

1. Tested at the minimum operating frequency of the part.
2. PEREQ input has an internal pull-down resistor.
3. BUSY# and ERROR# inputs each have an internal pull-up resistor.
4. I_{CC} max measurement at worse case frequency, V_{CC} and temperature ($0^{\circ}C$).
5. Not 100% tested.
6. I_{CC} HOT max measurement at worse case frequency, V_{CC} and max temperature.

The A.C. specifications given in Table 6.4 consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the CLK2 rising edge crossing the 2.0V level.

A.C. specification measurement is defined by Figure 6.1. Inputs must be driven to the voltage levels indicated by Figure 6.1 when A.C. specifications are measured. 80376 output delays are specified with minimum and maximum limits measured as shown. The minimum 80376 delay times are hold times provided to external circuitry. 80376 input setup and hold times are specified as minimums, defining the

smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 80376 processor operation.

Outputs NA#, W/R#, D/C#, M/IO#, LOCK#, BHE#, BLE#, A₂₃-A₁ and HLDA only change at the beginning of phase one. D₁₅-D₀ (write cycles) only change at the beginning of phase two. The READY#, HOLD, BUSY#, ERROR#, PEREQ and D₁₅-D₀ (read cycles) inputs are sampled at the beginning of phase one. The NA#, INTR and NMI inputs are sampled at the beginning of phase two.

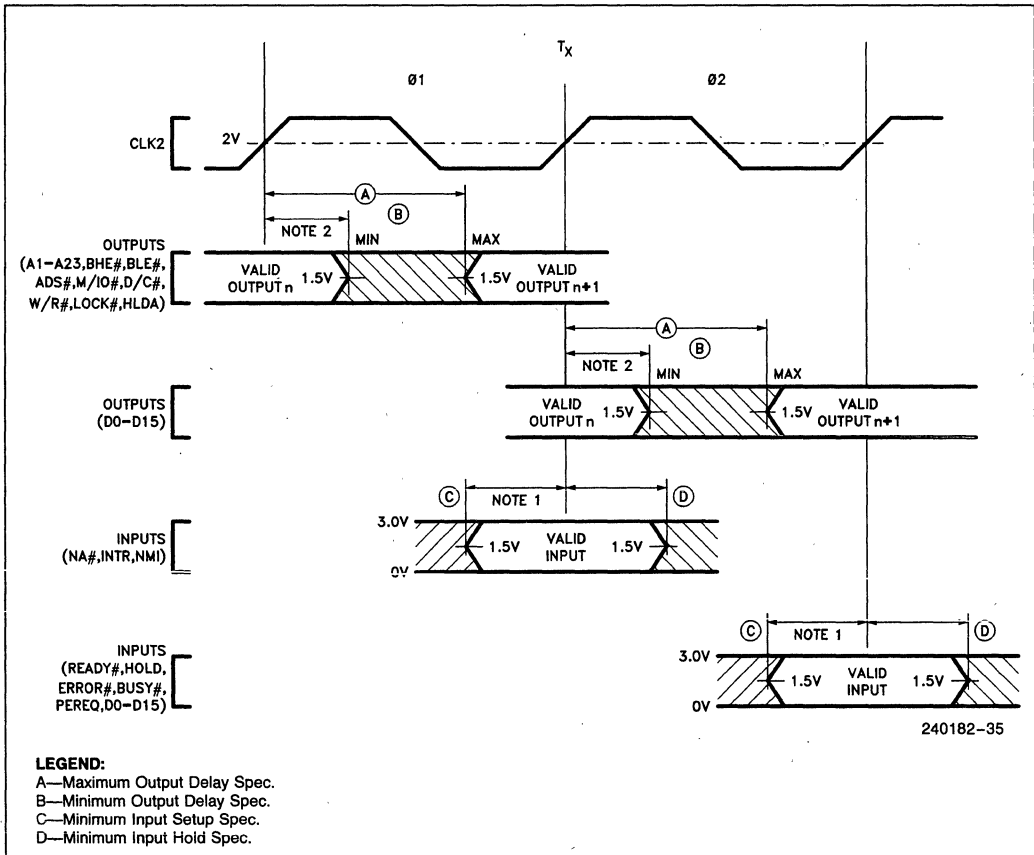


Figure 6.1. Drive Levels and Measurement Points for A.C. Specifications

6.4 A.C. Specifications

ADVANCE INFORMATION SUBJECT TO CHANGE

Table 6.4. 80376 A.C. Characteristics at 16 MHz

Functional Operating Range: $V_{CC} = 5V \pm 10\%$; $T_{CASE} = 0^{\circ}C$ to $115^{\circ}C$ for 88-pin PGA, $0^{\circ}C$ to $110^{\circ}C$ for 100-pin plastic

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Operating Frequency	4	16	MHz		Half CLK2 Freq
t_1	CLK2 Period	31	125	ns	6.3	*
t_{2a}	CLK2 HIGH Time	9		ns	6.3	At 2 ⁽³⁾
t_{2b}	CLK2 HIGH Time	5		ns	6.3	At $(V_{CC} - 0.8)V$ ⁽³⁾
t_{3a}	CLK2 LOW Time	9		ns	6.3	At 2V ⁽³⁾
t_{3b}	CLK2 LOW Time	7		ns	6.3	At 0.8V ⁽³⁾
t_4	CLK2 Fall Time		8	ns	6.3	$(V_{CC} - 0.8)V$ to 0.8V ⁽³⁾
t_5	CLK2 Rise Time		8	ns	6.3	0.8V to $(V_{CC} - 0.8)V$ ⁽³⁾
t_6	A ₂₃ -A ₁ Valid Delay	4	36	ns	6.5	$C_L = 120$ pF ⁽⁴⁾
t_7	A ₂₃ -A ₁ Float Delay	4	40	ns	6.6	(1)
t_8	BHE #, BLE #, LOCK # Valid Delay	4	36	ns	6.5	$C_L = 75$ pF ⁽⁴⁾
t_9	BHE #, BLE #, LOCK # Float Delay	4	40	ns	6.6	(1)
t_{10}	W/R #, M/IO #, D/C #, ADS # Valid Delay	6	33	ns	6.5	$C_L = 75$ pF ⁽⁴⁾
t_{11}	W/R #, M/IO #, D/C #, ADS # Float Delay	6	35	ns	6.6	(1)
t_{12}	D ₁₅ -D ₀ Write Data Valid Delay	4	40	ns	6.5	$C_L = 120$ pF ⁽⁴⁾
t_{13}	D ₁₅ -D ₀ Write Data Float Delay	4	35	ns	6.6	(1)
t_{14}	HLDA Valid Delay	6	33	ns	6.6	$C_L = 75$ pF ⁽⁴⁾
t_{15}	NA # Setup Time	5		ns	6.4	
t_{16}	NA # Hold Time	21		ns	6.6	
t_{19}	READY # Setup Time	19		ns	6.4	
t_{20}	READY # Hold Time	4		ns	6.4	
t_{21}	Setup Time D ₁₅ -D ₀ Read Data	9		ns	6.4	
t_{22}	Hold Time D ₁₅ -D ₀ Read Data	6		ns	6.4	
t_{23}	HOLD Setup Time	26		ns	6.4	
t_{24}	HOLD Hold Time	5		ns	6.4	
t_{25}	RESET Setup Time	13		ns	6.7	
t_{26}	RESET Hold Time	4		ns	6.7	

NOTE:

The 80376 does not have t_{17} or t_{18} timing specifications.

Table 6.4. 80376 A.C. Characteristics at 16 MHz

Functional Operating Range: $V_{CC} = 5V \pm 10\%$; $T_{CASE} = 0^{\circ}C$ to $115^{\circ}C$ for 80-pin PGA, $0^{\circ}C$ to $110^{\circ}C$ for 100-pin plastic (Continued)

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{27}	NMI, INTR Setup Time	16		ns	6.4	(2)
t_{28}	NMI, INTR Hold Time	16		ns	6.4	(2)
t_{29}	PEREQ, ERROR#, BUSY# Setup Time	16		ns	6.4	(2)
t_{30}	PEREQ, ERROR#, BUSY# Hold Time	5		ns	6.4	(2)

NOTES:

1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.
3. These are not tested. They are guaranteed by design characterization.
4. Tested with C_L set to 50 pF and derated to support the indicated distributed capacitive load. See Figure 6.8 for the capacitive derating curve.

A.C. TEST LOADS

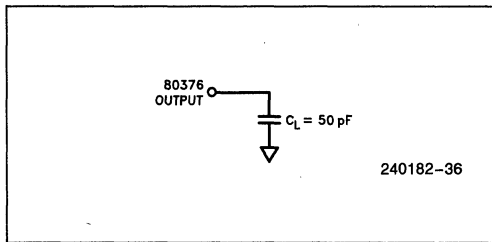


Figure 6.2. A.C. Test Loads

A.C. TIMING WAVEFORMS

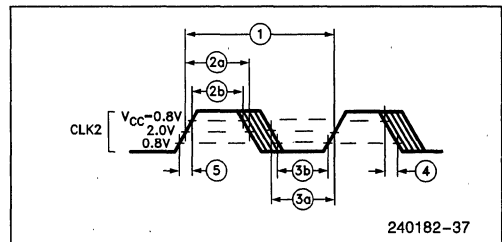


Figure 6.3. CLK2 Waveform

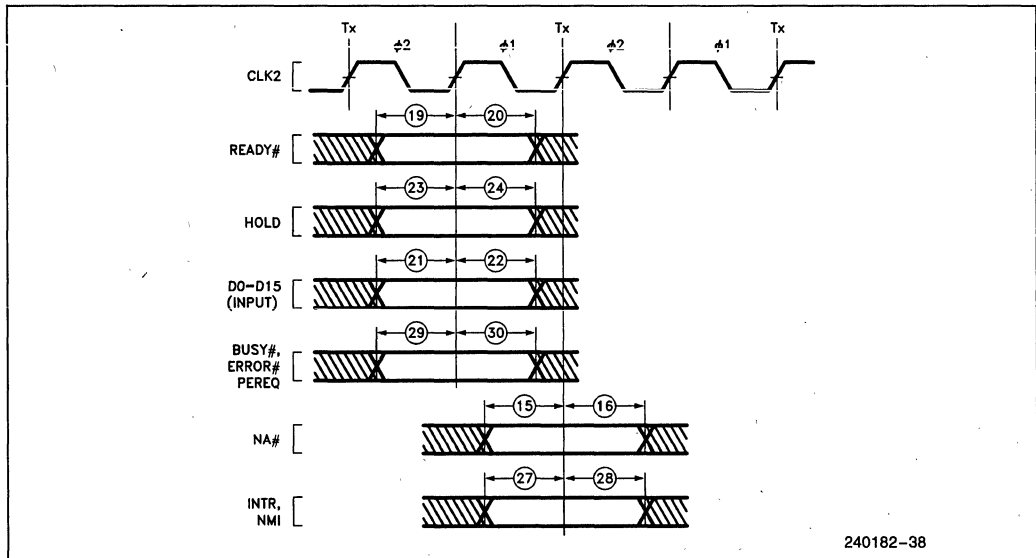


Figure 6.4. A.C. Timing Waveforms—Input Setup and Hold Timing

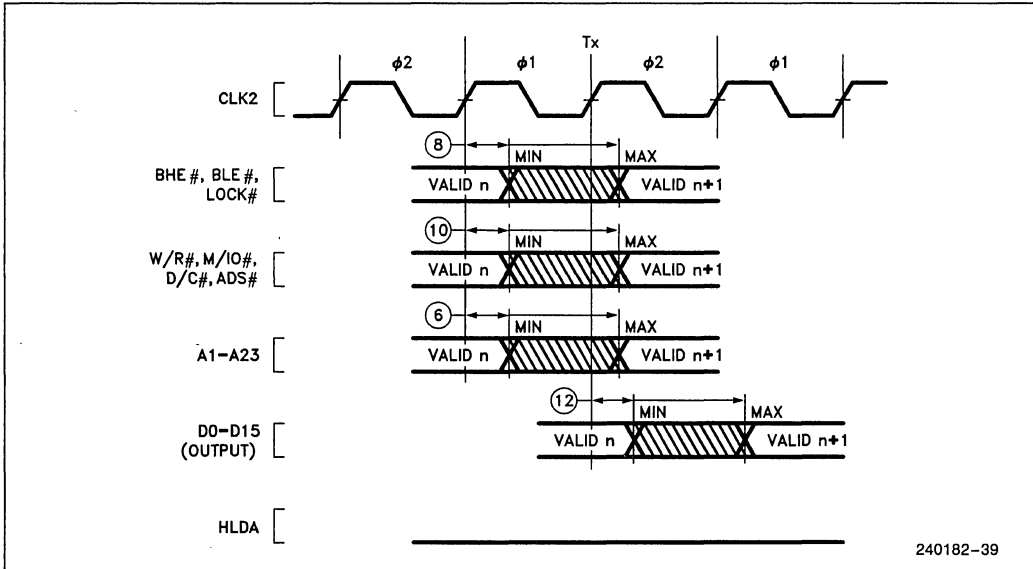


Figure 6.5. A.C. Timing Waveforms—Output Valid Delay Timing

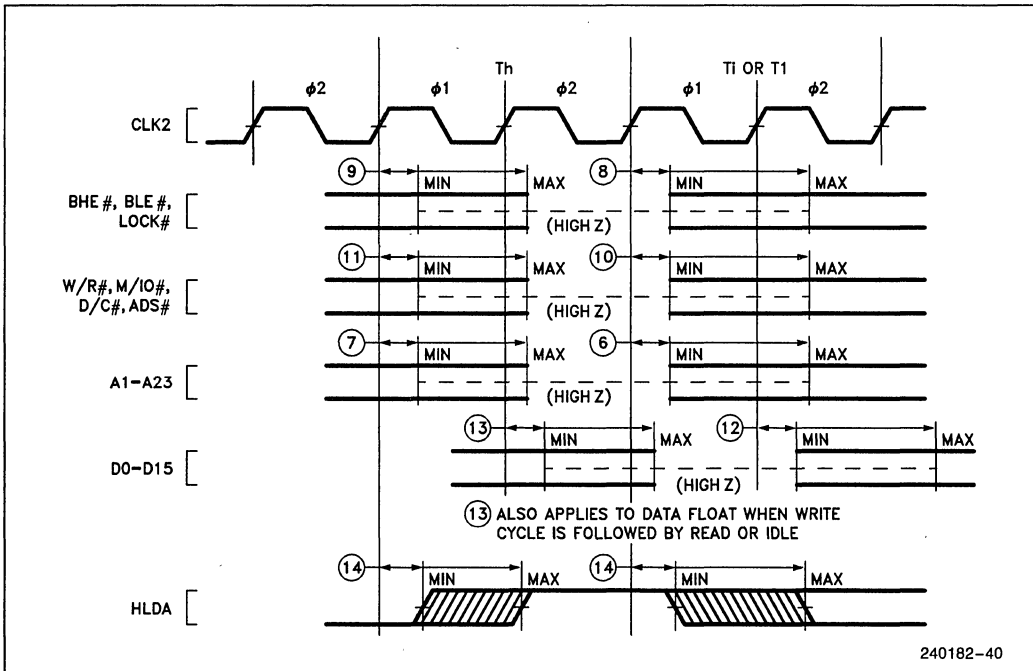


Figure 6.6. A.C. Timing Waveforms—Output Float Delay and HLDA Valid Delay Timing

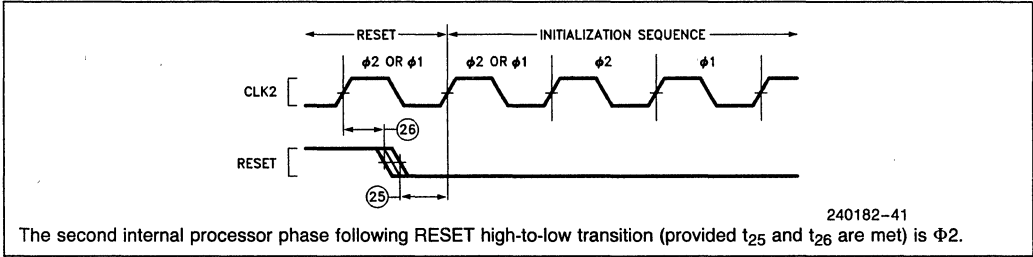


Figure 6.7. A.C. Timing Waveforms—RESET Setup and Hold Timing, and Internal Phase

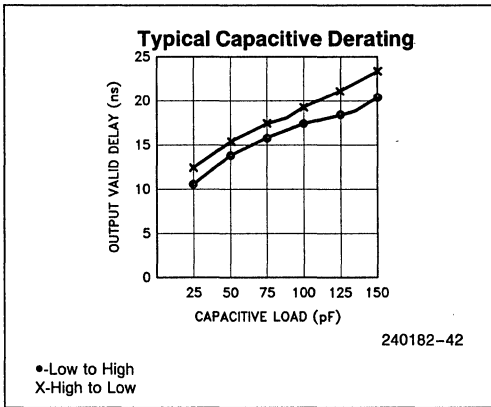


Figure 6.8. Capacitive Derating Curve

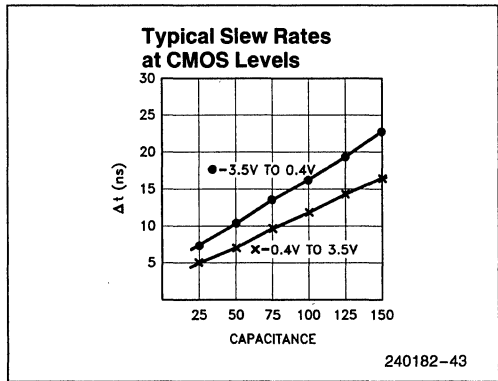


Figure 6.9. CMOS Level Slew Rates for Output Buffers

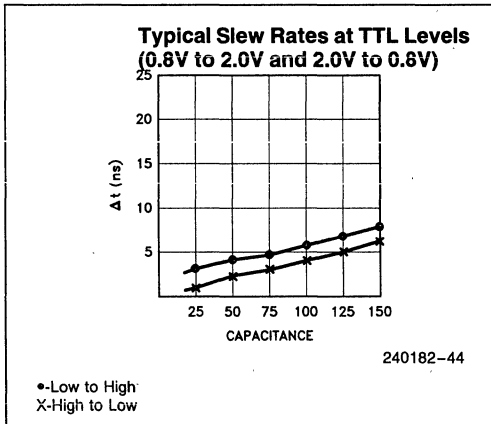


Figure 6.10. TTL Level Slew Rates for Output Buffers

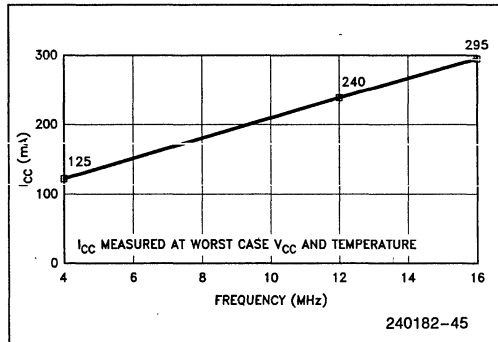


Figure 6.11. Typical I_{CC} vs Frequency

6.5 Designing for ICETM-376 Emulator (Advanced Data)

The 376 embedded processor in-circuit emulator product is the ICE-376 emulator. Use of the emulator requires the target system to provide a socket that is compatible with the ICE-376 emulator. The 80376 offers two different probes for emulating user systems: an 88-pin PGA probe and a 100-pin fine pitch flat-pack probe. The 100-pin fine pitch flat-pack probe requires a socket, called the 100-pin PQFP, which is available from 3-M text-tool (part number 2-0100-07243-000). The ICE-376 emulator probe attaches to the target system via an adapter which replaces the 80376 component in the target system. Because of the high operating frequency of 80376 systems and of the ICE-376 emulator, there is no buffering between the 80376 emulation processor in the ICE-376 emulator probe and the target system. A direct result of the non-buffered interconnect is that the ICE-376 emulator shares the address and data bus with the user's system, and the RESET signal is intercepted by the ICE emulator hardware. In order for the ICE-376 emulator to be functional in the user's system without the Optional Isolation Board (OIB) the designer must be aware of the following conditions:

1. The bus controller must only enable data transceivers onto the data bus during valid read cycles of the 80376, other local devices or other bus masters.
2. Before another bus master drives the local processor address bus, the other master must gain control of the address bus by asserting HOLD and receiving the HLDA response.

3. The emulation processor receives the RESET signal 2 or 4 CLK2 cycles later than an 80376 would, and responds to RESET later. Correct phase of the response is guaranteed.

In addition to the above considerations, the ICE-376 emulator processor module has several electrical and mechanical characteristics that should be taken into consideration when designing the 80376 system.

Capacitive Loading: ICE-376 adds up to 27 pF to each 80376 signal.

Drive Requirements: ICE-376 adds one FAST TTL load on the CLK2, control, address, and data lines. These loads are within the processor module and are driven by the 80376 emulation processor, which has standard drive and loading capability listed in Tables 6.3 and 6.4.

Power Requirements: For noise immunity and CMOS latch-up protection the ICE-376 emulator processor module is powered by the user system. The circuitry on the processor module draws up to 1.4A including the maximum 80376 I_{CC} from the user 80376 socket.

80376 Location and Orientation: The ICE-376 emulator processor module may require lateral clearance. Figure 6.12 shows the clearance requirements of the iMP adapter and Figure 6.13 shows the clearance requirements of the 88-pin PGA adapter. The

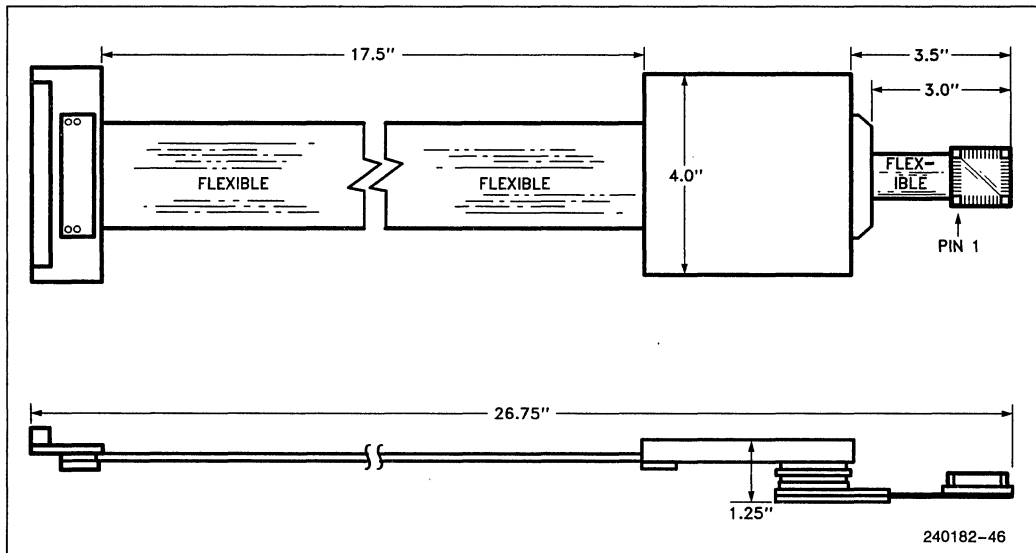


Figure 6.12. Preliminary ICETM-376 Emulator User Cable with PQFP Adapter

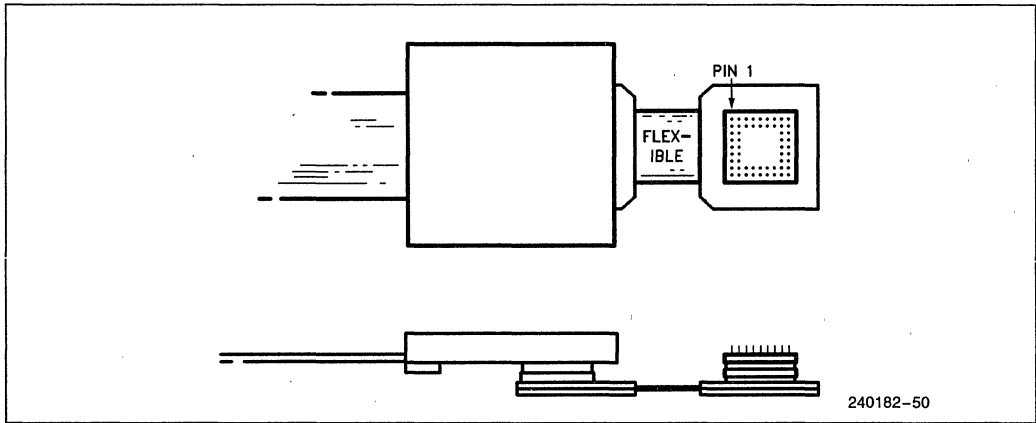


Figure 6.13. Preliminary ICETM-376 Emulator User Cable with 88-Pin PGA Adapter

optional isolation board (OIB), which provides extra electrical buffering and has the same lateral clearance requirements as Figures 6.12 and 6.13, adds an additional 0.5 inches to the vertical clearance requirement. This is illustrated in Figure 6.14.

on the user's bus. The OIB allows the ICE-376 emulator to function in user systems with faults (shorted signals, etc.). After electrical verification the OIB may be removed. When the OIB is installed, the user system must have a maximum CLK2 frequency of 20 MHz.

Optional Isolation Board (OIB) and the CLK2 speed reduction: Due to the unbuffered probe design, the ICE-376 emulator is susceptible to errors

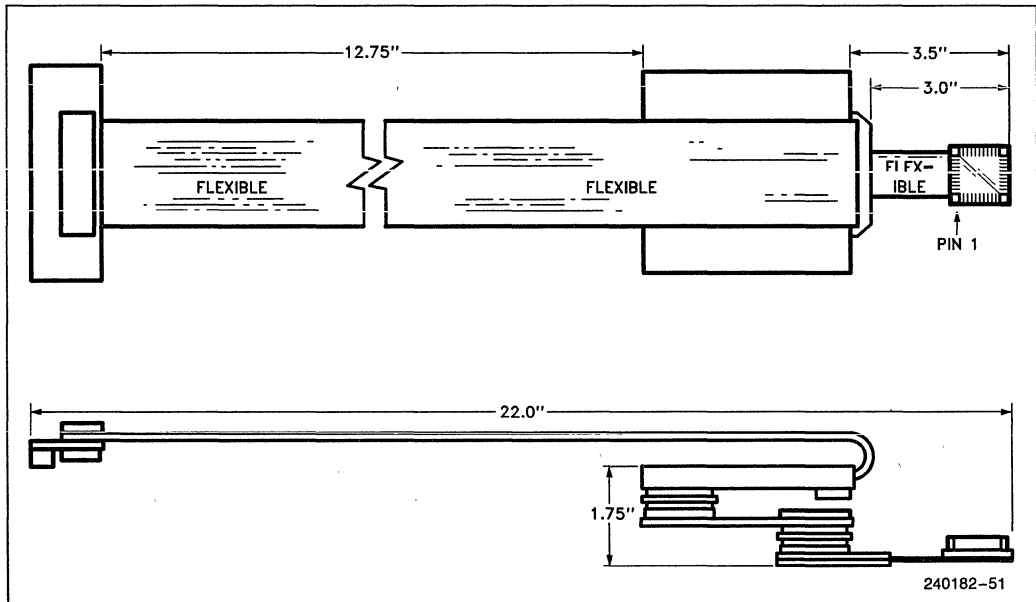


Figure 6.14. Preliminary ICETM-376 Emulator User Cable with OIB and PQFP Adapter

7.0 DIFFERENCES BETWEEN THE 80376 AND THE 80386

The following are the major differences between the 80376 and the 80386.

1. The 80376 generates byte selects on BHE# and BLE# (like the 8086 and 80286 microprocessors) to distinguish the upper and lower bytes on its 16-bit data bus. The 80386 uses four-byte selects, BE0#–BE3#, to distinguish between the different bytes on its 32-bit bus.
2. The 80376 has no bus sizing option. The 80386 can select between either a 32-bit bus or a 16-bit bus by use of the BS16# input. The 80376 has a 16-bit bus size.
3. The NA# pin operation in the 80376 is identical to that of the NA# pin on the 80386 with one exception: the NA# pin of the 80386 cannot be activated on 16-bit bus cycles (where BS16# is LOW in the 80386 case), whereas NA# can be activated on any 80376 bus cycle.
4. The contents of all 80376 registers at reset are identical to the contents of the 80386 registers at reset, except the DX register. The DX register contains a component-stepping identifier at reset, i.e.
 - in 80386, after reset DH = 3 indicates 80386
DL = revision number;
 - in 80376, after reset DH = 33H indicates 80376
DL = revision number.
5. The 80386 uses A₃₁ and M/IO# as a select for numerics coprocessor. The 80376 uses the A₂₃ and M/IO# to select its numerics coprocessor.
6. The 80386 prefetch unit fetches code in four-byte units. The 80376 prefetch unit reads two bytes as one unit (like the 80286 microprocessor). In BS16# mode, the 80386 takes two consecutive bus cycles to complete a prefetch request. If there is a data read or write request after the prefetch starts, the 80386 will fetch all four bytes before addressing the new request.
7. The 80376 has no paging mechanism.
8. The 80376 starts executing code in what corresponds to the 80386 protected mode. The 80386 starts execution in real mode, which is then used to enter protected mode.
9. The 80386 has a virtual-86 mode that allows the execution of a real mode 8086 program as a task in protected mode. The 80376 has no virtual-86 mode.
10. The 80386 maps a 48-bit logical address into a 32-bit physical address by segmentation and paging. The 80376 maps its 48-bit logical address into a 24-bit physical address by segmentation only.
11. The 80376 uses the 80387SX numerics coprocessor for floating point operations, while the 80386 uses the 80387 coprocessor.
12. The 80386 can execute from 16-bit code segments. The 80376 can **only** execute from 32-bit code Segments.

8.0 INSTRUCTION SET

This section describes the 376 embedded processor instruction set. Table 8.1 lists all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in the following sections, which completely describe the encoding structure and the definition of all fields occurring within 80376 instructions.

8.1 80376 Instruction Encoding and Clock Count Summary

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Table 8.1 below, by the processor clock period (e.g. 62.5 ns for an 80376 operating at 16 MHz). The actual clock count of an 80376 program will average 10% more

than the calculated clock count due to instruction sequences which execute faster than they can be fetched from memory.

Instruction Clock Count Assumptions:

1. The instruction has been prefetched, decoded, and is ready for execution.
2. Bus cycles do not require wait states.
3. There are no local bus HOLD requests delaying processor access to the bus.
4. No exceptions are detected during instruction execution.
5. If an effective address is calculated, it does not use two general register components. One register, scaling and displacement can be used within the clock counts shown. However, if the effective address calculation uses two general register components, add 1 clock to the clock count shown.
6. Memory reference instruction accesses byte or aligned 16-bit operands.

Instruction Clock Count Notation

- If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand.

—n = number of times repeated.

- m = number of components in the next instruction executed, where the entire displacement (if any) counts as one component, the entire immediate data (if any) counts as one component, and all other bytes of the instruction and prefix(es) each count as one component.

Misaligned or 32-Bit Operand Accesses:

- If instructions accesses a misaligned 16-bit operand or 32-bit operand on even address add:
 - 2* clocks for read or write.
 - 4** clocks for read and write.
- If instructions accesses a 32-bit operand on odd address add:
 - 4* clocks for read or write.
 - 8** clocks for read and write.

Wait States:

Wait states add 1 clock per wait state to instruction execution for each data access.

Table 8.1. 80376 Instruction Set Clock Count Summary

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
GENERAL DATA TRANSFER				
MOV = Move:				
Register to Register/Memory	1 000 1 00 w mod reg r/m	2/2*	0/1*	a
Register/Memory to Register	1 000 1 01 w mod reg r/m	2/4* *	0/1*	a
Immediate to Register/Memory	1 100 0 11 w mod 0 0 0 r/m immediate data	2/2*	0/1*	a
Immediate to Register (Short Form)	1 0 1 1 w reg immediate data	2	2	
Memory to Accumulator (Short Form)	1 0 1 0 0 0 0 w full displacement	4*	1*	a
Accumulator to Memory (Short Form)	1 0 1 0 0 0 1 w full displacement	2	1*	a
Register/Memory to Segment Register	1 000 1 11 0 mod sreg3 r/m	22/23	0/6*	a,b,c
Segment Register to Register/Memory	1 000 1 10 0 mod sreg3 r/m	2/2*	0/1*	a
MOVSX = Move with Sign Extension				
Register from Register/Memory	0 000 1 11 1 1 0 1 1 1 1 w mod reg r/m	3/6*	0/1*	a
MOVZX = Move with Zero Extension				
Register from Register/Memory	0 000 1 11 1 1 0 1 0 1 1 w mod reg r/m	3/6*	0/1*	a
PUSH = Push:				
Register/Memory	1 111 1 11 1 1 0 mod 1 1 0 r/m	7/9*	2/4*	a
Register (Short Form)	0 1 0 1 0 reg	4	2	a
Segment Register (ES, CS, SS or DS)	0 0 0 sreg2 1 1 0	4	2	a
Segment Register (FS or GS)	0 0 0 0 1 1 1 1 1 0 sreg3 0 0 0	4	2	a
Immediate	0 1 1 0 1 0 5 0 Immediate data	4	2	a
PUSHA = Push All				
	0 1 1 0 0 0 0 0	34	16	a
POP = Pop				
Register/Memory	1 000 1 11 1 1 0 mod 0 0 0 r/m	7/9*	2/4*	a
Register (Short Form)	0 1 0 1 1 reg	6	2	a
Segment Register (ES, SS or DS)	0 0 0 sreg2 1 1 1	25	6	a, b, c
Segment Register (FS or GS)	0 000 1 11 1 1 0 sreg3 0 0 1	25	6	a, b, c
POPA = Pop All				
	0 1 1 0 0 0 0 1	40	16	a
XCHG = Exchange				
Register/Memory with Register	1 000 0 11 w mod reg r/m	3/5**	0/2**	a, m
Register with Accumulator (Short Form)	1 0 0 1 0 reg	3	0	
IN = Input from:				
Fixed Port	1 1 1 0 0 1 0 w port number	6*	1*	f,k
		26*	1*	f,l
Variable Port	1 1 1 0 1 1 0 w	7*	1*	f,k
		27*	1*	f,l
OUT = Output to:				
Fixed Port	1 1 1 0 0 1 1 w port number	4*	1*	f,k
		24*	1*	f,l
Variable Port	1 1 1 0 1 1 1 w	5*	1*	f,k
		26*	1*	f,l
LEA = Load EA to Register				
	1 000 1 10 1 mod reg r/m	2		

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
SEGMENT CONTROL				
LDS = Load Pointer to DS	1 1 0 0 0 1 0 1 mod reg r/m	26*	6*	a, b, c
LES = Load Pointer to ES	1 1 0 0 0 1 0 0 mod reg r/m	26*	6*	a, b, c
LFS = Load Pointer to FS	0 0 0 0 1 1 1 1 1 0 1 1 0 1 0 0 mod reg r/m	29*	6*	a, b, c
LGS = Load Pointer to GS	0 0 0 0 1 1 1 1 1 0 1 1 0 1 0 1 mod reg r/m	29*	6*	a, b, c
LSS = Load Pointer to SS	0 0 0 0 1 1 1 1 1 0 1 1 0 0 1 0 mod reg r/m	26*	6*	a, b, c
FLAG CONTROL				
CLC = Clear Carry Flag	1 1 1 1 1 0 0 0	8		
CLD = Clear Direction Flag	1 1 1 1 1 1 0 0	8		
CLI = Clear Interrupt Enable Flag	1 1 1 1 1 0 1 0	8		f
CLTS = Clear Task Switched Flag	0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 0	6		e
CMC = Complement Carry Flag	1 1 1 1 0 1 0 1	2		
LAHF = Load AH into Flag	1 0 0 1 1 1 1 1	2		
POPF = Pop Flags	1 0 0 1 1 1 0 1	7		a, g
PUSHF = Push Flags	1 0 0 1 1 1 0 0	4		a
SAHF = Store AH into Flags	1 0 0 1 1 1 0 0	3		
STC = Set Carry Flag	1 1 1 1 1 0 0 1	2		
STD = Set Direction Flag	1 1 1 1 1 1 0 1	2		
STI = Set Interrupt Enable Flag	1 1 1 1 1 0 1 1	8		f
ARITHMETIC				
ADD = Add				
Register to Register *	0 0 0 0 0 d w mod reg r/m	2		
Register to Memory	0 0 0 0 0 0 w mod reg r/m	7**	2**	a
Memory to Register	0 0 0 0 0 1 w mod reg r/m	6*	1*	a
Immediate to Register/Memory	1 0 0 0 0 s w mod 0 0 0 r/m immediate data	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0 0 0 0 1 0 w immediate data	2		
ADC = Add with Carry				
Register to Register	0 0 0 1 0 d w mod reg r/m	2		
Register to Memory	0 0 0 1 0 0 w mod reg r/m	7**	2**	a
Memory to Register	0 0 0 1 0 0 1 w mod reg r/m	6*	1*	a
Immediate to Register/Memory	1 0 0 0 0 s w mod 0 1 0 r/m immediate data	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0 0 0 1 0 1 w immediate data	2		
INC = Increment				
Register/Memory	1 1 1 1 1 1 1 w mod 0 0 0 r/m	2/6**	0/2**	a
Register (Short Form)	0 1 0 0 reg	2		
SUB = Subtract				
Register from Register	0 0 1 0 1 0 d w mod reg r/m	2		

SEE INTEL FOR DESIGN-IN INFORMATION

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number Of Data Cycles	Notes
ARITHMETIC (Continued)				
Register from Memory	0010100w mod reg r/m	7**	2**	a
Memory from Register	0010101w mod reg r/m	6*	1	a
Immediate from Register/Memory	100000sw mod 101 r/m immediate data	2/7** *	0/1**	a
Immediate from Accumulator (Short Form)	0010110w immediate data	2		
SBB = Subtract with Borrow				
Register from Register	000110dw mod reg r/m	2		
Register from Memory	0001100w mod reg r/m	7**	2**	a
Memory from Register	0001101w mod reg r/m	6*	1*	a
Immediate from Register/Memory	100000sw mod 011 r/m immediate data	2/7**	0/2**	a
Immediate from Accumulator (Short Form)	0001110w immediate data	2		
DEC = Decrement				
Register/Memory	1111111w reg 001 r/m	2/6**	0/2**	a
Register (Short Form)	01001 reg	2		
CMP = Compare				
Register with Register	001110dw mod reg r/m	2		
Memory with Register	0011100w mod reg r/m	5*	1*	a
Register with Memory	0011101w mod reg r/m	6**	2**	a
Immediate with Register/Memory	100000sw mod 111 r/m immediate data	2/5*	0/1*	a
Immediate with Accumulator (Short Form)	0011110w immediate data	2		
NEG = Change Sign	1111011w mod 011 r/m	2/6*	0/2*	a
AAA = ASCII Adjust for Add	00110111	4		
AAS = ASCII Adjust for Subtract	00111111	4		
DAA = Decimal Adjust for Add	00100111	4		
DAS = Decimal Adjust for Subtract	00101111	4		
MUL = Multiply (Unsigned)				
Accumulator with Register/Memory	1111011w mod 100 r/m			
Multiplier—Byte		12–17/15–20	0/1	a,n
—Word		12–25/15–28*	0/1*	a,n
—Doubleword		12–41/17–46*	0/2*	a,n
IMUL = Integer Multiply (Signed)				
Accumulator with Register/Memory	1111011w mod 101 r/m			
Multiplier—Byte		12–17/15–20	0/1	a,n
—Word		12–25/15–28*	0/1*	a,n
—Doubleword		12–41/17–46*	0/2*	a,n
Register with Register/Memory	00001111 10101111 mod reg r/m			
Multiplier—Byte		12–17/15–20	0/1	a,n
—Word		12–25/15–28*	0/1*	a,n
—Doubleword		12–41/17–46*	0/2*	a,n
Register/Memory with Immediate to Register	011010s1 mod reg r/m immediate data			
—Word		13–26/14–27*	0/1*	a,n
—Doubleword		13–42/16–45*	0/2*	a,n

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number Of Data Cycles	Notes
ARITHMETIC (Continued)				
DIV = Divide (Unsigned)				
Accumulator by Register/Memory	1111011w mod110 r/m			
Divisor—Byte		17/17	0/1	a, o
—Word		22/25*	0/1*	a, o
—Doubleword		38/43*	0/2*	a, o
IDIV = Integer Divide (Signed)				
Accumulator by Register/Memory	1111011w mod111 r/m			
Divisor—Byte		19/22	0/1	a, o
—Word		27/30*	0/1	a, o
—Doubleword		43/48*	0/2*	a, o
AAD = ASCII Adjust for Divide	11010101 00001010	19		
AAM = ASCII Adjust for Multiply	11010100 00001010	17		
CBW = Convert Byte to Word	10011000	3		
CWD = Convert Word to Double Word	10011001	2		
LOGIC				
Shift Rotate Instructions Not Through Carry (ROL, ROR, SAL, SAR, SHL, and SHR)				
Register/Memory by 1	1101000w mod TTT r/m	3/7**	0/2**	a
Register/Memory by CL	1101001w mod TTT r/m	3/7**	0/2**	a
Register/Memory by Immediate Count	1100000w mod TTT r/m	3/7**	0/2**	a
Through Carry (RCL and RCR)				
Register/Memory by 1	1101000w mod TTT r/m	9/10**	0/2**	a
Register/Memory by CL	1101001w mod TTT r/m	9/10**	10/2**	a
Register/Memory by Immediate Count	1100000w mod TTT r/m	9/10**	0/2**	a
	TTT Instruction			
	000 ROL			
	001 ROR			
	010 RCL			
	011 RCR			
	100 SHL/SAL			
	101 SHR			
	111 SAR			
SHLD = Shift Left Double				
Register/Memory by Immediate	00001111 10100100 mod reg r/m	3/7**	0/2**	
Register/Memory by CL	00001111 10100101 mod reg r/m	3/7**	0/2**	
SHRD = Shift Right Double				
Register/Memory by Immediate	00001111 10101100 mod reg r/m	3/7**	0/2**	
Register/Memory by CL	00001111 10101101 mod reg r/m	3/7**	0/2**	
AND = And				
Register to Register	001000dw mod reg r/m	2		

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
LOGIC (Continued)				
Register to Memory	0010000w mod reg r/m	7*	2**	a
Memory to Register	0010001w mod reg r/m	6*	1*	a
Immediate to Register/Memory	1000000w mod 100 r/m	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0010010w	2		
TEST = And Function to Flags, No Result				
Register/Memory and Register	1000010w mod reg r/m	2/5*	0/1*	a
Immediate Data and Register/Memory	1111011w mod 000 r/m	2/5*	0/1*	a
Immediate Data and Accumulator (Short Form)	1010100w	2		
OR = Or				
Register to Register	000010dw mod reg r/m	2		
Register to Memory	0000100w mod reg r/m	7**	2**	a
Memory to Register	0000101w mod reg r/m	6*	1*	a
Immediate to Register/Memory	1000000w mod 001 r/m	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0000110w	2		
XOR = Exclusive Or				
Register to Register	0011000w mod reg r/m	2		
Register to Memory	0011000w mod reg r/m	7**	2**	a
Memory to Register	0011001w mod reg r/m	6*	1*	a
Immediate to Register/Memory	1000000w mod 110 r/m	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0011010w	2		
NOT = Invert Register/Memory	1111011w mod 010 r/m	2/6**	0/2**	a
STRING MANIPULATION				
CMPS = Compare Byte Word	1010011w	10*	2*	a
INS = Input Byte/Word from DX Port	0110110w	9**	1**	a,f,k
		29**	1**	a,f,l
LODS = Load Byte/Word to AL/AX/EAX	1010110w	5*	1*	a
MOVS = Move Byte Word	1010010w	7**	2**	a
OUTS = Output Byte/Word to DX Port	0110111w	8**	1**	a,f,k
		28**	1**	a,f,l
SCAS = Scan Byte Word	1010111w	7*	1*	a
STOS = Store Byte/Word from AL/AX/EX	1010101w	4*	1*	a
XLAT = Translate String	11010111	5*	1*	a
REPEATED STRING MANIPULATION				
Repeated by Count in CX or ECX				
REPE CMPS = Compare String (Find Non-Match)	11110011 1010011w	5 + 9n**	2n**	a

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes					
REPEATED STRING MANIPULATION (Continued)									
REPNE CMPS = Compare String (Find Match)	<table border="1"><tr><td>11110010</td><td>1010011w</td></tr></table>	11110010	1010011w	5 + 9n**	2n**	a			
11110010	1010011w								
REP INS = Input String	<table border="1"><tr><td>11110011</td><td>0110110w</td></tr></table>	11110011	0110110w	7 + 5n* 27 + 6n*	1n* 1n*	a,f,k a,f,l			
11110011	0110110w								
REP LODS = Load String	<table border="1"><tr><td>11110011</td><td>1010110w</td></tr></table>	11110011	1010110w	6 + 6n*	1n*	a			
11110011	1010110w								
REP MOVS = Move String	<table border="1"><tr><td>11110011</td><td>1010010w</td></tr></table>	11110011	1010010w	7 + 4n**	2n**	a			
11110011	1010010w								
REP OUTS = Output String	<table border="1"><tr><td>11110011</td><td>0110111w</td></tr></table>	11110011	0110111w	6 + 5n* 26 + 5n*	1n* 1n*	a,f,k a,f,l			
11110011	0110111w								
REPE SCAS = Scan String (Find Non-AL/AX/EAX)	<table border="1"><tr><td>11110011</td><td>1010111w</td></tr></table>	11110011	1010111w	5 + 8n*	1n*	a			
11110011	1010111w								
REPNE SCAS = Scan String (Find AL/AX/EAX)	<table border="1"><tr><td>11110010</td><td>1010111w</td></tr></table>	11110010	1010111w	5 + 8n*	1n*	a			
11110010	1010111w								
REP STOS = Store String	<table border="1"><tr><td>11110011</td><td>1010101w</td></tr></table>	11110011	1010101w	5 + 5n*	1n*	a			
11110011	1010101w								
BIT MANIPULATION									
BSF = Scan Bit Forward	<table border="1"><tr><td>00001111</td><td>10111100</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	10111100	mod reg	r/m	10 + 3n**	2n**	a	
00001111	10111100	mod reg	r/m						
BSR = Scan Bit Reverse	<table border="1"><tr><td>00001111</td><td>10111101</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	10111101	mod reg	r/m	10 + 3n**	2n**	a	
00001111	10111101	mod reg	r/m						
BT = Test Bit									
Register/Memory, Immediate	<table border="1"><tr><td>00001111</td><td>10111010</td><td>mod 00</td><td>r/m</td><td>immed 8-bit data</td></tr></table>	00001111	10111010	mod 00	r/m	immed 8-bit data	3/6*	0/1*	a
00001111	10111010	mod 00	r/m	immed 8-bit data					
Register/Memory, Register	<table border="1"><tr><td>00001111</td><td>10100011</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	10100011	mod reg	r/m	3/12*	0/1*	a	
00001111	10100011	mod reg	r/m						
BTC = Test Bit and Complement									
Register/Memory, Immediate	<table border="1"><tr><td>00001111</td><td>10111010</td><td>mod 111</td><td>r/m</td><td>immed 8-bit data</td></tr></table>	00001111	10111010	mod 111	r/m	immed 8-bit data	6/8*	0/2*	a
00001111	10111010	mod 111	r/m	immed 8-bit data					
Register/Memory, Register	<table border="1"><tr><td>00001111</td><td>10111011</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	10111011	mod reg	r/m	6/13*	0/2*	a	
00001111	10111011	mod reg	r/m						
BTR = Test Bit and Reset									
Register/Memory, Immediate	<table border="1"><tr><td>00001111</td><td>10111010</td><td>mod 110</td><td>r/m</td><td>immed 8-bit data</td></tr></table>	00001111	10111010	mod 110	r/m	immed 8-bit data	6/8*	0/2*	a
00001111	10111010	mod 110	r/m	immed 8-bit data					
Register/Memory, Register	<table border="1"><tr><td>00001111</td><td>10110011</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	10110011	mod reg	r/m	6/13*	0/2*	a	
00001111	10110011	mod reg	r/m						
BTS = Test Bit and Set									
Register/Memory, Immediate	<table border="1"><tr><td>00001111</td><td>10111010</td><td>mod 101</td><td>r/m</td><td>immed 8-bit data</td></tr></table>	00001111	10111010	mod 101	r/m	immed 8-bit data	6/8*	0/2*	a
00001111	10111010	mod 101	r/m	immed 8-bit data					
Register/Memory, Register	<table border="1"><tr><td>00001111</td><td>10101011</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	10101011	mod reg	r/m	6/13*	0/2*	a	
00001111	10101011	mod reg	r/m						
CONTROL TRANSFER									
CALL = Call									
Direct within Segment	<table border="1"><tr><td>11110100</td><td>full displacement</td></tr></table>	11110100	full displacement	9 + m*	2	j			
11110100	full displacement								
Register/Memory									
Indirect within Segment	<table border="1"><tr><td>11111111</td><td>mod 010</td><td>r/m</td></tr></table>	11111111	mod 010	r/m	9 + m/12 + m	2/3	a,j		
11111111	mod 010	r/m							
Direct Intersegment	<table border="1"><tr><td>10011010</td><td>unsigned full offset, selector</td></tr></table>	10011010	unsigned full offset, selector	42 + m	9	c,d,j			
10011010	unsigned full offset, selector								

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
CONTROL TRANSFER (Continued)				
(Direct Intersegment)				
Via Call Gate to Same Privilege Level		64 + m	13	a,c,d,j
Via Call Gate to Different Privilege Level, (No Parameters)		98 + m	13	a,c,d,j
Via Call Gate to Different Privilege Level, (x Parameters)		106 + 8x + m	13 + 4x	a,c,d,j
From 386 Task to 386 TSS		392	124	a,c,d,j
Indirect Intersegment	1 1 1 1 1 1 1 1 mod 0 1 1 r/m	46 + m	10	a,c,d,j
Via Call Gate to Same Privilege Level		68 + m	14	a,c,d,j
Via Call Gate to Different Privilege Level, (No Parameters)		102 + m	14	a,c,d,j
Via Call Gate to Different Privilege Level, (x Parameters)		110 + 8x + m	14 + 4x	a,c,d,j
From 386 Task to 386 TSS		399	130	a,c,d,j
JMP = Unconditional Jump				
Short	1 1 1 0 1 0 1 1 8-bit displacement	7 + m		j
Direct within Segment	1 1 1 0 1 0 0 1 full displacement	7 + m		j
Register/Memory Indirect within Segment	1 1 1 1 1 1 1 1 mod 1 0 0 r/m	9 + m/14 + m	2/4	a,j
Direct Intersegment	1 1 1 0 1 0 1 0 unsigned full offset, selector	37 + m	5	c,d,j
Via Call Gate to Same Privilege Level		53 + m	9	a,c,d,j
From 386 Task to 386 TSS		395	124	a,c,d,j
Indirect Intersegment	1 1 1 1 1 1 1 1 mod 1 0 1 r/m	37 + m	9	a,c,d,j
Via Call Gate to Same Privilege Level		59 + m	13	a,c,d,j
From 386 Task to 386 TSS		401	124	a,c,d,j

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
CONTROL TRANSFER (Continued)				
RET = Return from CALL:				
Within Segment	1 1 0 0 0 0 1 1	12 + m	2	a,j,p
Within Segment Adding Immediate to SP	1 1 0 0 0 0 1 0 16-bit displ	12 + m*	2	a,j,p
Intersegment	1 1 0 0 1 0 1 1	12	4	a,c,d,j,p
Intersegment Adding Immediate to SP	1 1 0 0 1 0 1 0 16-bit displ	12 + m	4	a,c,d,j,p
to Different Privilege Level				
Intersegment		12	4	c,d,j,p
Intersegment Adding Immediate to SP		12 + m	4	c,d,j,p
CONDITIONAL JUMPS				
NOTE: Times Are Jump "Taken or Not Taken"				
JO = Jump on Overflow				
8-Bit Displacement	0 1 1 1 0 0 0 0 8-bit displ	7 + m or 3		j
Full Displacement	0 0 0 0 1 1 1 1 10 0 0 0 0 0 0 full displacement	7 + m or 3		j
JNO = Jump on Not Overflow				
8-Bit Displacement	0 1 1 1 0 0 0 1 8-bit displ	7 + m or 3		j
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 full displacement	7 + m or 3		j
JB/JNAE = Jump on Below/Not Above or Equal				
8-Bit Displacement	0 0 1 1 0 0 1 0 8-bit displ	7 + m or 3		j
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 full displacement	7 + m or 3		j
JNB/JAE = Jump on Not Below/Above or Equal				
8-Bit Displacement	0 1 1 1 0 0 1 1 8-bit displ	7 + m or 3		j
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 full displacement	7 + m or 3		j
JE/JZ = Jump on Equal/Zero				
8-Bit Displacement	0 1 1 1 0 1 0 0 8-bit displ	7 + m or 3		j
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 full displacement	7 + m or 3		j
JNE/JNZ = Jump on Not Equal/Not Zero				
8-Bit Displacement	0 1 1 1 0 1 0 1 8-bit displ	7 + m or 3		j
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 1 full displacement	7 + m or 3		j
JBE/JNA = Jump on Below or Equal/Not Above				
8-Bit Displacement	0 1 1 1 0 1 1 0 8-bit displ	7 + m or 3		j
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 0 full displacement	7 + m or 3		j
JNBE/JA = Jump on Not Below or Equal/Above				
8-Bit Displacement	0 1 1 1 0 1 1 1 8-bit displ	7 + m or 3		j
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 full displacement	7 + m or 3		j
JS = Jump on Sign				
8-Bit Displacement	0 1 1 1 1 0 0 0 8-bit displ	7 + m or 3		j
Full Displacement	0 0 0 0 1 1 1 1 1 0 0 0 1 0 0 0 full displacement	7 + m or 3		j

SEE ADVANCE INFORMATION FOR DESIGN-IN INFORMATION

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
CONDITIONAL JUMPS (Continued)				
JNS = Jump on Not Sign				
8-Bit Displacement	01111001 8-bit displ	7 + m or 3		j
Full Displacement	00001111 10001001 full displacement	7 + m or 3		j
JP/JPE = Jump on Parity/Parity Even				
8-Bit Displacement	01111010 8-bit displ	7 + m or 3		j
Full Displacement	00001111 10001010 full displacement	7 + m or 3		j
JNP/JPO = Jump on Not Parity/Parity Odd				
8-Bit Displacement	01111011 8-bit displ	7 + m or 3		j
Full Displacement	00001111 10001011 full displacement	7 + m or 3		j
JL/JNGE = Jump on Less/Not Greater or Equal				
8-Bit Displacement	01111100 8-bit displ	7 + m or 3		j
Full Displacement	00001111 10001100 full displacement	7 + m or 3		j
JNL/JGE = Jump on Not Less/Greater or Equal				
8-Bit Displacement	01111101 8-bit displ	7 + m or 3		j
Full Displacement	00001111 10001101 full displacement	7 + m or 3		j
JLE/JNG = Jump on Less or Equal/Not Greater				
8-Bit Displacement	01111110 8-bit displ	7 + m or 3		j
Full Displacement	00001111 10001110 full displacement	7 + m or 3		j
JNLE/JG = Jump on Not Less or Equal/Greater				
8-Bit Displacement	01111111 8-bit displ	7 + m or 3		j
Full Displacement	00001111 10001111 full displacement	7 + m or 3		j
JECXZ = Jump on ECX Zero				
	11000011 8-bit displ	9 + m or 5		j
(Address Size Prefix Differentiates JECXZ from JECXZ)				
LOOP = Loop ECX Times				
	11100010 8-bit displ	11 + m		j
LOOPZ/LOOPE = Loop with Zero/Equal				
	11100001 8-bit displ	11 + m		j
LOOPNZ/LOOPNE = Loop While Not Zero				
	11100000 8-bit displ	11 + m		j
CONDITIONAL BYTE SET				
NOTE: Times Are Register/Memory				
SETO = Set Byte on Overflow				
To Register/Memory	00001111 10010000 mod 000 r/m	4/5*	0/1*	a
SETNO = Set Byte on Not Overflow				
To Register/Memory	00001111 10010001 mod 000 r/m	4/5*	0/1*	a
SETB/SETNAE = Set Byte on Below/Not Above or Equal				
To Register/Memory	00001111 10010010 mod 000 r/m	4/5*	0/1*	a

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
CONDITIONAL BYTE SET (Continued)				
SETNB = Set Byte on Not Below/Above or Equal				
To Register/Memory	00001111 10010011 mod000 r/m	4/5*	0/1*	a
SETE/SETZ = Set Byte on Equal/Zero				
To Register/Memory	00001111 10010100 mod000 r/m	4/5*	0/1*	a
SETNE/SETNZ = Set Byte on Not Equal/Not Zero				
To Register/Memory	00001111 10010101 mod000 r/m	4/5*	0/1*	a
SETBE/SETNA = Set Byte on Below or Equal/Not Above				
To Register/Memory	00001111 10010110 mod000 r/m	4/5*	0/1*	a
SETNBE/SETA = Set Byte on Not Below or Equal/Above				
To Register/Memory	00001111 10010111 mod000 r/m	4/5*	0/1*	a
SETS = Set Byte on Sign				
To Register/Memory	00001111 10011000 mod000 r/m	4/5*	0/1*	a
SETNS = Set Byte on Not Sign				
To Register/Memory	00001111 10011001 mod000 r/m	4/5*	0/1*	a
SETP/SETPE = Set Byte on Parity/Parity Even				
To Register/Memory	00001111 10011010 mod000 r/m	4/5*	0/1*	a
SETNP/SETPO = Set Byte on Not Parity/Parity Odd				
To Register/Memory	00001111 10011011 mod000 r/m	4/5*	0/1*	a
SETL/SETNGE = Set Byte on Less/Not Greater or Equal				
To Register/Memory	00001111 01011000 mod000 r/m	4/5*	0/1*	a
SETNL/SETGE = Set Byte on Not Less/Greater or Equal				
To Register/Memory	00001111 01011001 mod000 r/m	4/5*	0/1*	a
SETLE/SETNG = Set Byte on Less or Equal/Not Greater				
To Register/Memory	00001111 10011110 mod000 r/m	4/5*	0/1*	a
SETNLE/SETG = Set Byte on Not Less or Equal/Greater				
To Register/Memory	00001111 10011111 mod000 r/m	4/5*	0/1*	a
ENTER = Enter Procedure	11001000 16-bit displacement, 8-bit level			
L = 0		10		a
L = 1		14	1	a
L > 1		17 + 8(n - 1)	4(n - 1)	a
LEAVE = Leave Procedure	11001001	6		a

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts*	Number of Data Cycles	Notes
INTERRUPT INSTRUCTIONS				
INT = Interrupt:				
Type Specified	11001101 type			
Via Interrupt or Trap Gate to Same Privilege Level		71	14	c,d,i,p
Via Interrupt or Trap Gate to Different Privilege Level		111	14	c,d,i,p
From 386 Task to 386 TSS via Task Gate		467	140	c,d,i,p
Type 3				
	11001100			
Via Interrupt or Trap Gate to Same Privilege Level		71	14	c,d,i,p
Via Interrupt or Trap Gate to Different Privilege Level		111	14	c,d,i,p
From 386 Task to 386 TSS via Task Gate		308	138	c,d,i,p
INTO = Interrupt 4 if Overflow Flag Set				
	11001110			
If OF = 1:		3		
If OF = 0:				
Via Interrupt or Trap Gate to Same Privilege Level		71	14	c,d,i,p
Via Interrupt or Trap Gate to Different Privilege Level		111	14	c,d,i,p
From 386 Task to 386 TSS via Task Gate		413	138	c,d,i,p

ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN
INFORMATION

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number Of Data Cycles	Notes
INTERRUPT INSTRUCTIONS (Continued)				
Bound = Out of Range Interrupt 5 If Detect Value	0 1 1 0 0 0 1 0 mod reg r/m			
If In Range		19	0	a,c,d,j,o,p
If Out of Range: Via Interrupt or Trap Gate to Same Privilege Level		14	14	c,d,j,p
Via Interrupt or Trap Gate to Different Privilege Level		14	14	c,d,j,p
From 386 Task to 386 TSS via Task Gate		398	138	c,d,j,p
INTERRUPT RETURN				
IRET = Interrupt Return	1 1 0 0 1 1 1 1			
To the Same Privilege Level (within Task)		42	5	a,c,d,j,p
To Different Privilege Level (within Task)		86	5	a,c,d,j,p
From 386 Task to 386 TSS		328	138	c,d,j,p
PROCESSOR CONTROL				
HLT = HALT	1 1 1 1 0 1 0 0 0	5		b
MOV = Move to and from Control/Debug/Test Registers				
CR0 from register	0 0 0 0 1 1 1 1 0 0 1 0 0 1 0 1 1 eee reg	10		b
Register from CR0	0 0 0 0 1 1 1 1 0 0 1 0 0 0 0 0 1 1 eee reg	6		b
DR0-3 from Register	0 0 0 0 1 1 1 1 0 0 1 0 0 0 1 1 1 1 eee reg	22		b
DR6-7 from Register	0 0 0 0 1 1 1 1 0 0 1 0 0 0 1 1 1 1 eee reg	16		b
Register from DR6-7	0 0 0 0 1 1 1 1 0 0 1 0 0 0 0 1 1 1 eee reg	14		b
Register from DR0-3	0 0 0 0 1 1 1 1 0 0 1 0 0 0 0 1 1 1 eee reg	22		b
NOP = No Operation	1 0 0 1 0 0 0 0	3		
WAIT = Wait until BUSY # Pin is Negated	1 0 0 1 1 0 1 1	6		

ADVANCE INFORMATION *
 SEE INTEL FOR DESIGN-IN INFORMATION

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
PROCESSOR EXTENSION INSTRUCTIONS				
Processor Extension Escape	11011TTT mod LLL r/m TTT and LLL bits are opcode information for coprocessor.	See 80387SX Data Sheet		a
PREFIX BYTES				
Address Size Prefix	01100111	0		
LOCK = Bus Lock Prefix	11110000	0		f
Operand Size Prefix	01100110	0		
Segment Override Prefix				
CS:	00101110	0		
DS:	00111110	0		
ES:	00100110	0		
FS:	01100100	0		
GS:	01100101	0		
SS:	00110110	0		
PROTECTION CONTROL				
ARPL = Adjust Requested Privilege Level				
From Register/Memory	01f00001 mod reg r/m	20/21**	2**	a
LAR = Load Access Rights				
From Register/Memory	00001111 00000010 mod reg r/m	17/18*	1*	a,c,i,p
LGDT = Load Global Descriptor				
Table Register	00001111 00000001 mod 010 r/m	13**	3*	a,e
LIDT = Load Interrupt Descriptor				
Table Register	00001111 00000001 mod 011 r/m	13**	3*	a,e
LLDT = Load Local Descriptor				
Table Register to Register/Memory	00001111 00000000 mod 010 r/m	24/28*	5*	a,c,e,p
LMSW = Load Machine Status Word				
From Register/Memory	00001111 00000001 mod 110 r/m	10/13*	1*	a,e
LSL = Load Segment Limit				
From Register/Memory	00001111 00000011 mod reg r/m			
Byte-Granular Limit		24/27*	2*	a,c,i,p
Page-Granular Limit		29/32*	2*	a,c,i,p
LTR = Load Task Register				
From Register/Memory	00001111 00000000 mod 001 r/m	27/31*	4*	a,c,e,p
SGDT = Store Global Descriptor				
Table Register	00001111 00000001 mod 000 r/m	11*	3*	a
SIDT = Store Interrupt Descriptor				
Table Register	00001111 00000001 mod 001 r/m	11*	3*	a
SLDT = Store Local Descriptor Table Register				
To Register/Memory	00001111 00000000 mod 000 r/m	2/2*	4*	a

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
PROTECTION CONTROL (Continued)				
SMSW = Store Machine Status Word	0 0001 111 00000001 mod 100 r/m	2/2*	1*	a, c
STR = Store Task Register To Register/Memory	0 0001 111 00000000 mod 001 r/m	2/2*	1*	a
VERR = Verify Read Access Register/Memory	0 0001 111 00000000 mod 100 r/m	10/11**	2**	a,c,i,p
VERW = Verify Write Access	0 0001 111 00000000 mod 101 r/m	15/16**	2**	a,c,i,p

NOTES:

- a. Exception 13 fault (general violation) will occur if the memory operand in CS, DS, ES, FS or GS cannot be used due to either a segment limit violation or access rights violation. If a stack limit is violated, and exception 12 (stack segment limit violation or not present) occurs.
- b. For segment load operations, the CPL, RPL and DPL must agree with the privilege rules to avoid an exception 13 fault (general protection violation). The segments's descriptor must indicate "present" or exception 11 (CS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 (stack segment limit violation or not present occurs).
- c. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert LOCK# to maintain descriptor integrity in multiprocessor systems.
- d. JMP, CALL, INT, RET and IRET instructions referring to another code segment will cause an exception 13 (general protection violation) if an applicable privilege rule is violated.
- e. An exception 13 fault occurs if CPL is greater than 0.
- f. An exception 13 fault occurs if CPL is greater than IOPL.
- g. The IF bit of the flag register is not updated if CPL is greater than IOPL. The IOPL field of the flag register is updated only if CPL = 0.
- h. Any violation of privilege rules as applied to the selector operand does not cause a protection exception; rather, the zero flag is cleared.
- i. If the coprocessor's memory operand violates a segment limit or segment access rights, an exception 13 fault (general protection exception) will occur before the ESC instruction is executed. An exception 12 fault (stack segment limit violation or no present) will occur if the stack limit is violated by the operand's starting address.
- j. The destination of a JMP, CALL, INT, RET or IRET must be in the defined limit of a code segment or an exception 13 fault (general protection violation) will occur.
- k. If $CPL \leq IOPL$
- l. If $CPL > IOPL$
- m. LOCK# is automatically asserted, regardless of the presence or absence of the LOCK# prefix.
- n. The 80376 uses an early-out multiply algorithm. The actual number of clocks depends on the position of the most significant bit in the operand (multiplier). Clock counts given are minimum to maximum. To calculate actual clocks use the following formula:

$$\text{Actual Clock} = \text{if } m < > 0 \text{ then } \max([\log_2 |m|], 3) + 9 \text{ clocks:}$$

$$\text{if } m = 0 \text{ then } 12 \text{ clocks (where } m \text{ is the multiplier)}$$
- o. An exception may occur, depending on the value of the operand.
- p. LOCK# is asserted during descriptor table accesses.

8.2 INSTRUCTION ENCODING

Overview

All instruction encodings are subsets of the general instruction format shown in Figure 8.1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the "mod r/m" byte and "scaled index" byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain

encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 8.1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 8.2 is a complete list of all fields appearing in the 80376 instruction set. Further ahead, following Table 8.2, are detailed tables for each field.

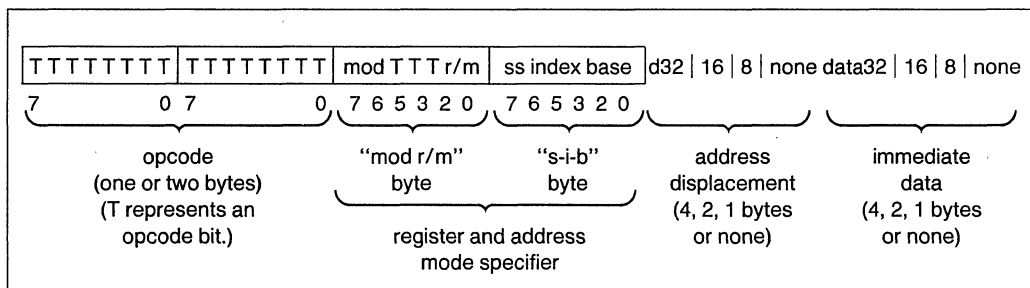


Figure 8.1. General Instruction Format

Table 8.2. Fields within 80376 Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 Bits)	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field Must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
tttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

Note: Table 8.1 shows encoding of individual instructions.

16-Bit Extensions of the Instruction Set

Two prefixes, the operand size prefix (66H) and the effective address size prefix (67H), allow overriding individually the default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the opcode bytes. The presence of the operand size prefix (66H) and the effective address prefix will allow 16-bit data operation and 16-bit effective address calculations.

For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

Encoding of Instruction Fields

Within the instruction are several fields indicating register selection, addressing mode and so on.

ENCODING OF OPERAND LENGTH (w) FIELD

For any given instruction performing a data operation, the instruction will execute as a 32-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size with 66H Prefix	Normal Operand Size
0	8 Bits	8 Bits
1	16 Bits	32 Bits

ENCODING OF THE GENERAL REGISTER (reg) FIELD

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the "mod r/m" byte, or as the r/m field of the "mod r/m" byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected with 66H Prefix	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field with 66H Prefix		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field without 66H Prefix		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

ENCODING OF THE SEGMENT REGISTER (sreg) FIELD

The sreg field in certain instructions is a 2-bit field allowing one of the CS, DS, ES or SS segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the FS and GS segment registers to be specified also.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

ENCODING OF ADDRESS MODE

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the "mod r/m" byte, and a second byte of addressing information, the "s-i-b" (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the "mod r/m" byte has r/m = 100 and mod = 00, 01 or 10. When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the "mod r/m" byte, also contains three bits (shown as TTT in Figure 8.1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the "mod r/m" byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the "mod r/m" byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following three pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Encoding of Normal Address Mode with "mod r/m" byte (no "s-i-b" byte present):

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by reg or r/m during Normal Data Operations:		
mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Register Specified by reg or r/m during 16-Bit Data Operations: (66H Prefix)		
mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Encoding of 16-bit Address Mode with "mod r/m" Byte Using 67H Prefix

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	SS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Encoding of 32-bit Address Mode ("mod r/m" byte and "s-i-b" byte present):

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

****IMPORTANT NOTE:**

When index field is 100, indicating "no index register," then ss field MUST equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

NOTE:

Mod field in "mod r/m" byte; ss, index, base fields in "s-i-b" byte.

**ENCODING OF OPERATION
DIRECTION (d) FIELD**

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory <- - Register "reg" Field Indicates Source Operand; "mod r/m" or "mod ss index base" Indicates Destination Operand
1	Register <- - Register/Memory "reg" Field Indicates Destination Operand; "mod r/m" or "mod ss index base" Indicates Source Operand

ENCODING OF SIGN-EXTEND (s) FIELD

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16 32
0	None	None
1	Sign-Extend Data8 to Fill 16-Bit or 32-Bit Destination	None

**ENCODING OF CONDITIONAL
TEST (ttn) FIELD**

For the conditional instructions (conditional jumps and set on condition), ttn is encoded with n indicating to use the condition ($n=0$) or its negation ($n=1$), and ttt giving the condition to test.

Mnemonic	Condition	ttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

**ENCODING OF CONTROL OR DEBUG
REGISTER (eee) FIELD**

For the loading and storing of the Control and Debug registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	Reserved
011	Reserved
Do not use any other encoding	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

9.0 REVISION HISTORY

This 80376 data sheet, version -002, contains updates and improvements to previous versions. A revision summary is listed here for your convenience.

The sections significantly revised since version -001 are:

Front Page	The 80376 Microarchitecture diagram was added.
Section 1.0	Figure 1.2 was updated to show both top and bottom views of the 88-pin PGA package.
Section 2.0	Figure 2.0 was updated to show the 16-bit registers SI, DI, BP and SP.
Section 2.1	Figure 2.2 was updated to show the correct bit polarity for bit 4 in the CR0 register.
Section 2.1	Tables 2.1 and 2.2 were updated to include additional information on the EFLAGS and CR0 registers.
Section 2.3	Figure 2.3 was updated to more accurately reflect the addressing mechanism of the 80376.
Section 2.6	In the subsection Maskable Interrupt a paragraph was added to describe the effect of interrupt gates on the IF EFLAGS bit.
Section 2.8	Table 2.7 was updated to reflect the correct power up condition of the CR0 register.
Section 2.10	Figure 2.6 was updated to show the correct bit positions of the BT, BS and BD bits in the DR6 register.
Section 3.0	Figure 3.1 was updated to clearly show the address calculation process.
Section 3.2	The subsection DESCRIPTORS was elaborated upon to clearly define the relationship between the linear address space and physical address space of the 80376.
Section 3.2	Figures 3.3 and 3.4 were updated to show the AVL bit field.
Section 3.3	The last sentence in the first paragraph of subsection PROTECTION AND I/O PERMISSION BIT MAP was deleted. This was an incorrect statement.
Section 4.1	In the Subsection ADDRESS BUS (BHE#, BLE#, A₂₃-A₁) last sentence in the first paragraph was updated to reflect the numerics operand addresses as 8000FCH and 8000FEH. Because the 80376 sometimes does a double word I/O access a second access to 8000FEH can be seen.
Section 4.1	The Subsection Hold Latencies was updated to describe how 32-bit and unaligned accesses are internally locked but do not assert the LOCK# signal.
Section 4.2	Table 4.6 was updated to show the correct active data bits during a BLE# assertion.
Section 4.4	This section was updated to correctly reflect the pipelining of the address and status of the 80376 as opposed to "Address Pipelining" which occurs on processors such as the 80286.
Section 4.6	Table 4.7 was updated to show the correct Revision number, 05H.
Section 4.7	Table 4.8 was updated to show the numerics operand register 8000FEH. This address is seen when the 80376 does a DWORD operation to the port address 8000FCH.
Section 5.0	In the first paragraph the case temperatures were updated to correctly reflect the 0°C–115°C for the ceramic package and 0°C–110°C for the plastic package.
Section 6.2	Table 6.2 was updated to correctly reflect the Case Temperature under Bias specification of –65°C–120°C.
Section 6.4	Figure 6.8 vertical axis was updated to reflect "Output Valid Delay (ns)".
Section 6.4	Figure 6.11 was updated to show typical I _{CC} vs Frequency for the 80376.
Section 6.5	This entire section was updated to reflect the new ICE-376 emulator.
Section 8.1	The clock counts and opcodes for various instructions were updated to their correct value.
Section 8.2	The section INSTRUCTION ENCODING was appended to the data sheet.

The sections significantly revised since version -002 are:

Section 1.0	Modified table 1.1. to list pins in alphabetical order.
-------------	---

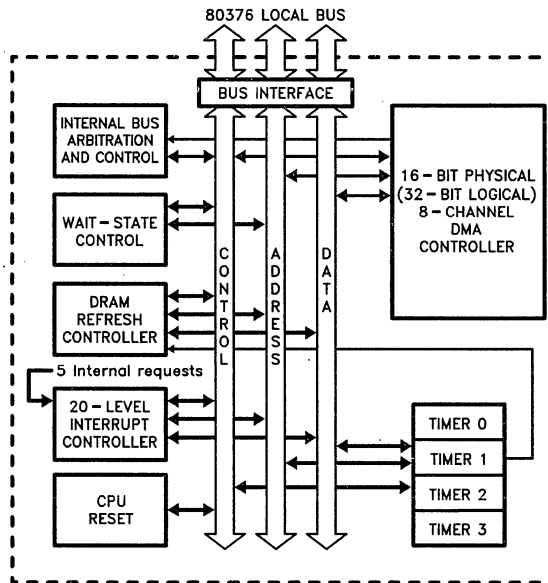


82370 INTEGRATED SYSTEM PERIPHERAL

- **High Performance 32-Bit DMA Controller for 16-Bit Bus**
 - 16 MBytes/Sec Maximum Data Transfer Rate at 16 MHz
 - 8 Independently Programmable Channels
- **20-Source Interrupt Controller**
 - Individually Programmable Interrupt Vectors
 - 15 External, 5 Internal Interrupts
 - 82C59A Superset
- **Four 16-Bit Programmable Interval Timers**
 - 82C54 Compatible
- **Software Compatible to 82380**
- **Programmable Wait State Generator**
 - 0 to 15 Wait States Pipelined
 - 1 to 16 Wait States Non-Pipelined
- **DRAM Refresh Controller**
- **80376 Shutdown Detect and Reset Control**
 - Software/Hardware Reset
- **High Speed CHMOS III Technology**
- **100-Pin Plastic Quad Flat-Pack Package and 132-Pin Pin Grid Array Package**
(See Packaging Handbook Order # 231369)
- **Optimized for Use with the 80376 Microprocessor**
 - Resides on Local Bus for Maximum Bus Bandwidth

The 82370 is a multi-function support peripheral that integrates system functions necessary in an 80376 environment. It has eight channels of high performance 32-bit DMA (32-bit internal, 16-bit external) with the most efficient transfer rates possible on the 80376 bus. System support peripherals integrated into the 82370 provide Interrupt Control, Timers, Wait State generation, DRAM Refresh Control, and System Reset logic.

The 82370's DMA Controller can transfer data between devices of different data path widths using a single channel. Each DMA channel operates independently in any of several modes. Each channel has a temporary data storage register for handling non-aligned data without the need for external alignment logic.



Internal Block Diagram

290164-1

Pin Descriptions

The 82370 provides all of the signals necessary to interface an 80376 host processor. It has a separate 24-bit address and 16-bit data bus. It also has a set of control signals to support operation as a bus master or a bus slave. Several special function signals

exist on the 82370 for interfacing the system support peripherals to their respective system counterparts. Following are the definitions of the individual pins of the 82370. These brief descriptions are provided as a reference. Each signal is further defined within the sections which describe the associated 82370 function.

Symbol	Type	Name and Function
A ₁ -A ₂₃	I/O	ADDRESS BUS: Outputs physical memory or port I/O addresses. See Address Bus (2.2.3) for additional information.
BHE# BLE#	I/O	BYTE ENABLES: Indicate which data bytes of the data bus take part in a bus cycle. See Byte Enable (2.2.4) for additional information.
D ₀ -D ₁₅	I/O	DATA BUS: This is the 16-bit data bus. These pins are active outputs during interrupt acknowledges, during Slave accesses, and when the 82370 is in the Master Mode.
CLK2	I	PROCESSOR CLOCK: This pin must be connected to the processor's clock, CLK2. The 82370 monitors the phase of this clock in order to remain synchronized with the CPU. This clock drives all of the internal synchronous circuitry.
D/C#	I/O	DATA/CONTROL: D/C# is used to distinguish between CPU control cycles and DMA or CPU data access cycles. It is active as an output only in the Master Mode.
W/R#	I/O	WRITE/READ: W/R# is used to distinguish between write and read cycles. It is active as an output only in the Master Mode.
M/IO#	I/O	MEMORY/IO: M/IO# is used to distinguish between memory and IO accesses. It is active as an output only in the Master Mode.
ADS#	I/O	ADDRESS STATUS: This signal indicates presence of a valid address on the address bus. It is active as output only in the Master Mode. ADS# is active during the first T-state where addresses and control signals are valid.
NA#	I	NEXT ADDRESS: Asserted by a peripheral or memory to begin a pipelined address cycle. This pin is monitored only while the 82370 is in the Master Mode. In the Slave Mode, pipelining is determined by the current and past status of the ADS# and READY# signals.
HOLD	O	HOLD REQUEST: This is an active-high signal to the Bus Master to request control of the system bus. When control is granted, the Bus Master activates the hold acknowledge signal (HLDA).
HLDA	I	HOLD ACKNOWLEDGE: This input signal tells the DMA controller that the Bus Master has relinquished control of the system bus to the DMA controller.

Pin Descriptions (Continued)

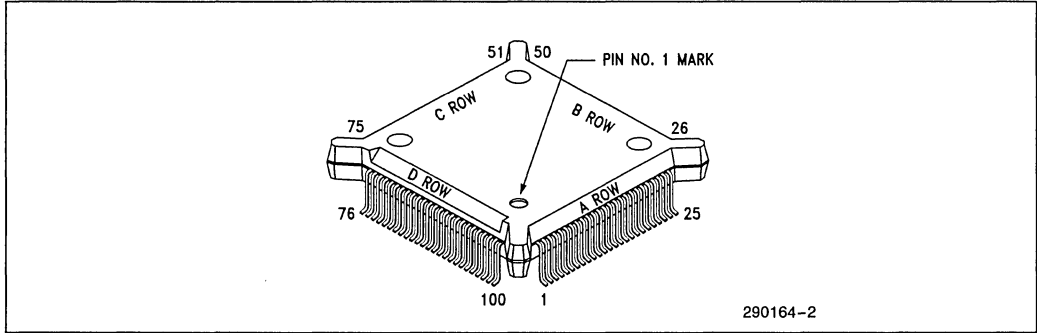
Symbol	Type	Name and Function
DREQ (0–3, 5–7)	I	DMA REQUEST: The DMA Request inputs monitor requests from peripherals requiring DMA service. Each of the eight DMA channels has one DREQ input. These active-high inputs are internally synchronized and prioritized. Upon request, channel 0 has the highest priority and channel 7 the lowest.
DREQ4/IRQ9 #	I	<p>DMA/INTERRUPT REQUEST: This is the DMA request input for channel 4. It is also connected to the interrupt controller via interrupt request 9. This internal connection is available for DMA channel 4 only. The interrupt input is active low and can be programmed as either edge or level triggered. Either function can be masked by the appropriate mask register. Priorities of the DMA channel and the interrupt request are not related but follow the rules of the individual controllers.</p> <p>Note that this pin has a weak internal pull-up. This causes the interrupt request to be inactive, but the DMA request will be active if there is no external connection made. Most applications will require that either one or the other of these functions be used, but not both. For this reason, it is advised that DMA channel 4 be used for transfers where a software request is more appropriate (such as memory-to-memory transfers). In such an application, DREQ4 can be masked by software, freeing IRQ9 # for other purposes.</p>
EOP #	I/O	<p>END OF PROCESS: As an output, this signal indicates that the current Requester access is the last access of the currently operating DMA channel. It is activated when Terminal Count is reached. As an input, it signals the DMA channel to terminate the current buffer and proceed to the next buffer, if one is available. This signal may be programmed as an asynchronous or synchronous input.</p> <p>EOP # must be connected to a pull-up resistor. This will prevent erroneous external requests for termination of a DMA process.</p>
EDACK (0–2)	O	ENCODED DMA ACKNOWLEDGE: These signals contain the encoded acknowledgment of a request for DMA service by a peripheral. The binary code formed by the three signals indicates which channel is active. Channel 4 does not have a DMA acknowledge. The inactive state is indicated by the code 100. During a Requester access, EDACK presents the code for the active DMA channel. During a Target access, EDACK presents the inactive code 100.
IRQ (11–23) #	I	INTERRUPT REQUEST: These are active low interrupt request inputs. The inputs can be programmed to be edge or level sensitive. Interrupt priorities are programmable as either fixed or rotating. These inputs have weak internal pull-up resistors. Unused interrupt request inputs should be tied inactive externally.
INT	O	INTERRUPT OUT: INT signals that an interrupt request is pending.
CLKIN	I	TIMER CLOCK INPUT: This is the clock input signal to all of the 82370's programmable timers. It is independent of the system clock input (CLK2).
TOUT1/REF #	O	TIMER 1 OUTPUT/REFRESH: This pin is software programmable as either the direct output of Timer 1, or as the indicator of a refresh cycle in progress. As REF #, this signal is active during the memory read cycle which occurs during refresh.

Pin Descriptions (Continued)

Symbol	Type	Name and Function
TOUT2# /IRQ3#	I/O	TIMER 2 OUTPUT/INTERRUPT REQUEST: This is the inverted output of Timer 2. It is also connected directly to interrupt request 3. External hardware can use IRQ3# if Timer 2 is programmed as OUT = 0 (TOUT2# = 1).
TOUT3#	O	TIMER 3 OUTPUT: This is the inverted output of Timer 3.
READY#	I	READY INPUT: This active-low input indicates to the 82370 that the current bus cycle is complete. READY is sampled by the 82370 both while it is in the Master Mode, and while it is in the Slave Mode.
WSC (0-1)	I	WAIT STATE CONTROL: WSC0 and WSC1 are inputs used by the Wait-State Generator to determine the number of wait states required by the currently accessed memory or I/O. The binary code on these pins, combined with the M/IO# signal, selects an internal register in which a wait-state count is stored. The combination WSC = 11 disables the wait-state generator.
READYO#	O	READY OUTPUT: This is the synchronized output of the wait-state generator. It is also valid during CPU accesses to the 82370 in the Slave Mode when the 82370 requires wait states. READYO# should feed directly the processor's READY# input.
RESET	I	RESET: This synchronous input serves to initialize the state of the 82370 and provides basis for the CPURST output. RESET must be held active for at least 15 CLK2 cycles in order to guarantee the state of the 82370. After Reset, the 82370 is in the Slave Mode with all outputs except timers and interrupts in their inactive states. The state of the timers and interrupt controller must be initialized through software. This input must be active for the entire time required by the host processor to guarantee proper reset.
CHPSEL#	O	CHIP SELECT: This pin is driven active whenever the 82370 is addressed in a slave bus read or write cycle. It is also active during interrupt acknowledge cycles when the 82370 is driving the Data Bus. It can be used to control the local bus transceivers to prevent contention with the system bus.
CPURST	O	CPU RESET: CPURST provides a synchronized reset signal for the CPU. It is activated in the event of a software reset command, a processor shut-down detect, or a hardware reset via the RESET pin. The 82370 holds CPURST active for 62 clocks in response to either a software reset command or a shut-down detection. Otherwise CPURST reflects the RESET input.
V _{CC}		POWER: +5V input power.
V _{SS}		Ground Reference.

Table 1. Wait-State Select Inputs

Port Address	Wait-State Registers				Select Inputs	
	D7	D4	D3	D0	WSC1	WSC0
72H	MEMORY 0		I/O 0		0	0
73H	MEMORY 1		I/O 1		0	1
74H	MEMORY 2		I/O 2		1	0
	DISABLED				1	1
M/IO#	1		0			



100 Pin Quad Flat-Pack Pin Out (Top View)

A Row		B Row		C Row		D Row	
Pin	Label	Pin	Label	Pin	Label	Pin	Label
1	CPURST	26	V _{CC}	51	A ₁₁	76	DREQ5
2	INT	27	D ₁₁	52	A ₁₀	77	DREQ4/IRQ9#
3	V _{CC}	28	D ₄	53	A ₉	78	DREQ3
4	V _{SS}	29	D ₁₂	54	A ₈	79	DREQ2
5	TOUT2#/IRQ3#	30	D ₅	55	A ₇	80	DREQ1
6	TOUT3#	31	D ₁₃	56	A ₆	81	DREQ0
7	D/C#	32	D ₆	57	A ₅	82	IRQ23#
8	V _{CC}	33	V _{SS}	58	V _{CC}	83	IRQ22#
9	W/R#	34	D ₁₄	59	A ₄	84	IRQ21#
10	M/IO#	35	D ₇	60	A ₃	85	IRQ20#
11	HOLD	36	D ₁₅	61	A ₂	86	IRQ19#
12	TOUT1/REF#	37	A ₂₃	62	A ₁	87	IRQ18#
13	CLK2	38	A ₂₂	63	V _{SS}	88	IRQ17#
14	V _{SS}	39	A ₂₁	64	BLE#	89	IRQ16#
15	READYO#	40	A ₂₀	65	BHE#	90	IRQ15#
16	EOP#	41	A ₁₉	66	V _{SS}	91	IRQ14#
17	CHPSEL#	42	A ₁₈	67	ADS#	92	IRQ13#
18	V _{CC}	43	V _{CC}	68	V _{CC}	93	IRQ12#
19	D ₀	44	A ₁₇	69	EDACK2	94	IRQ11#
20	D ₈	45	A ₁₆	70	EDACK1	95	CLKIN
21	D ₁	46	A ₁₅	71	EDACK0	96	WSC0
22	D ₉	47	A ₁₄	72	HLDA	97	WSC1
23	D ₂	48	V _{SS}	73	DREQ7	98	RESET
24	D ₁₀	49	A ₁₃	74	DREQ6	99	READY#
25	D ₃	50	A ₁₂	75	NA#	100	V _{SS}

	A	B	C	D	E	F	G	H	J	K	L	M	N	P								
1	V _{SS} ○	V _{CC} ○	V _{SS} ○	V _{CC} ○	A12 ○	A9 ○	A8 ○	A5 ○	A3 ○	BHE# ○	DREQ0 ○	EDACK1 ○	V _{SS} ○	V _{CC} ○								
2	V _{CC} ○	A19 ○	A17 ○	A15 ○	A13 ○	A10 ○	A7 ○	A4 ○	A1 ○	ADS# ○	EDACK2 ○	INT ○	V _{SS} ○	V _{CC} ○								
3	V _{SS} ○	A21 ○	A18 ○	A16 ○	A14 ○	A11 ○	A6 ○	A2 ○	BLE# ○	DREQ4/ IRQ9# ○	EDACK0 ○	HLDA ○	DREQ7 ○	DREQ5 ○								
4	V _{CC} ○	A22 ○	A20 ○	BOTTOM VIEW METAL LID (82370)								DREQ6 ○	NA# ○	DREQ3 ○								
5	(NC) ○	(NC) ○	A23 ○									WSC0 ○	DREQ2 ○	DREQ1 ○								
6	(NC) ○	(NC) ○	(NC) ○									WSC1 ○	IRQ22# ○	IRQ23# ○								
7	(NC) ○	(NC) ○	(NC) ○									IRQ21# ○	IRQ20# ○	IRQ19# ○								
8	(NC) ○	(NC) ○	D15 ○									IRQ17# ○	IRQ16# ○	IRQ18# ○								
9	D7 ○	(NC) ○	(NC) ○									IRQ13# ○	IRQ14# ○	IRQ15# ○								
10	D14 ○	D6 ○	D13 ○									D/C# ○	IRQ12# ○	IRQ11# ○								
11	(NC) ○	D5 ○	(NC) ○									READY# ○	CLKIN ○	W/R# ○								
12	V _{CC} ○	(NC) ○	D12 ○									(NC) ○	D3 ○	D10 ○	(NC) ○	READY# ○	HOLD ○	CHPSEL# ○	EOP# ○	CPURST ○	RESET ○	V _{CC} ○
13	V _{SS} ○	(NC) ○	D4 ○									(NC) ○	(NC) ○	D2 ○	D9 ○	(NC) ○	(NC) ○	TOUT1/ REF# ○	M/IO# ○	TOUT3# ○	TOUT2#/ IRQ5 ○	V _{SS} ○
14	V _{CC} ○	V _{SS} ○	V _{CC} ○									D11 ○	(NC) ○	(NC) ○	CLK2 ○	D1 ○	D0 ○	D8 ○	V _{SS} ○	V _{CC} ○	V _{SS} ○	V _{CC} ○

290164-3

82370 PGA Pinout

Pin	Label	Pin	Label	Pin	Label	Pin	Label
G14	CLK2	D14	D ₁₁	L1	DREQ0	A2	V _{CC}
N12	RESET	F12	D ₁₀	P6	IRQ23#	P2	V _{CC}
M12	CPURST	G13	D ₉	N6	IRQ22#	A4	V _{CC}
C5	A ₂₃	K14	D ₈	M7	IRQ21#	A12	V _{CC}
B4	A ₂₂	A9	D ₇	N7	IRQ20#	P12	V _{CC}
B3	A ₂₁	B10	D ₆	P7	IRQ19#	A14	V _{CC}
C4	A ₂₀	B11	D ₅	P8	IRQ18#	C14	V _{CC}
B2	A ₁₉	C13	D ₄	M8	IRQ17#	M14	V _{CC}
C3	A ₁₈	E12	D ₃	N8	IRQ16#	P14	V _{CC}
C2	A ₁₇	F13	D ₂	P9	IRQ15#	A5	NC
D3	A ₁₆	H14	D ₁	N9	IRQ14#	B5	NC
D2	A ₁₅	J14	D ₀	M9	IRQ13#	A6	NC
E3	A ₁₄	P11	W/R#	N10	IRQ12#	B6	NC
E2	A ₁₃	L13	M/IO#	P10	IRQ11#	C6	NC
E1	A ₁₂	K2	ADS#	M5	WSC0	A7	NC
F3	A ₁₁	M10	D/C#	M6	WSC1	B7	NC
F2	A ₁₀	N4	NA#	M13	TOUT3#	C7	NC
F1	A ₉	M11	READY#	N13	TOUT2#/IRQ3#	A8	NC
G1	A ₈	H12	READYO#	K13	TOUT1/REF#	B8	NC
G2	A ₇	J12	HOLD	N11	CLKIN	B9	NC
G3	A ₆	M3	HLDA	A1	V _{SS}	C9	NC
H1	A ₅	M2	INT	C1	V _{SS}	A11	NC
H2	A ₄	L12	EOP#	N1	V _{SS}	B12	NC
J1	A ₃	L2	EDACK2	N2	V _{SS}	C11	NC
H3	A ₂	M1	EDACK1	A3	V _{SS}	D12	NC
J2	A ₁	L3	EDACK0	A13	V _{SS}	G12	NC
J3	BLE#	N3	DREQ7	P13	V _{SS}	B13	NC
K1	BHE#	M4	DREQ6	B14	V _{SS}	D13	NC
K12	CHPSEL#	P3	DREQ5	L14	V _{SS}	E13	NC
C8	D ₁₅	K3	DREQ4/IRQ9#	N14	V _{SS}	H13	NC
A10	D ₁₄	P4	DREQ3	B1	V _{CC}	J13	NC
C10	D ₁₃	N5	DREQ2	D1	V _{CC}	E14	NC
C12	D ₁₂	P5	DREQ1	P1	V _{CC}	F14	NC

1.0 FUNCTIONAL OVERVIEW

The 82370 contains several independent functional modules. The following is a brief discussion of the components and features of the 82370. Each module has a corresponding detailed section later in this data sheet. Those sections should be referred to for design and programming information.

1.1 82370 Architecture

The 82370 is comprised of several computer system functions that are normally found in separate LSI and VLSI components. These include: a high-performance, eight-channel, 32-bit Direct Memory Access Controller; a 20-level Programmable Interrupt

Controller which is a superset of the 82C59A; four 16-bit Programmable Interval Timers which are functionally equivalent to the 82C54 timers; a DRAM Refresh Controller; a Programmable Wait State Generator; and system reset logic. The interface to the 82370 is optimized for high-performance operation with the 80376 microprocessor.

The 82370 operates directly on the 80376 bus. In the Slave Mode, it monitors the state of the processor at all times and acts or idles according to the commands of the host. It monitors the address pipeline status and generates the programmed number of wait states for the device being accessed. The 82370 also has logic to the reset of the 80376 via hardware or software reset requests and processor shutdown status.

After a system reset, the 82370 is in the Slave Mode. It appears to the system as an I/O device. It becomes a bus master when it is performing DMA transfers.

To maintain compatibility with existing software, the registers within the 82370 are accessed as bytes. If the internal logic of the 82370 requires a delay before another access by the processor, wait states

are automatically inserted into the access cycle. This allows the programmer to write initialization routines, etc. without regard to hardware recovery times.

Figure 1-1 shows the basic architectural components of the 82370. The following sections briefly discuss the architecture and function of each of the distinct sections of the 82370.

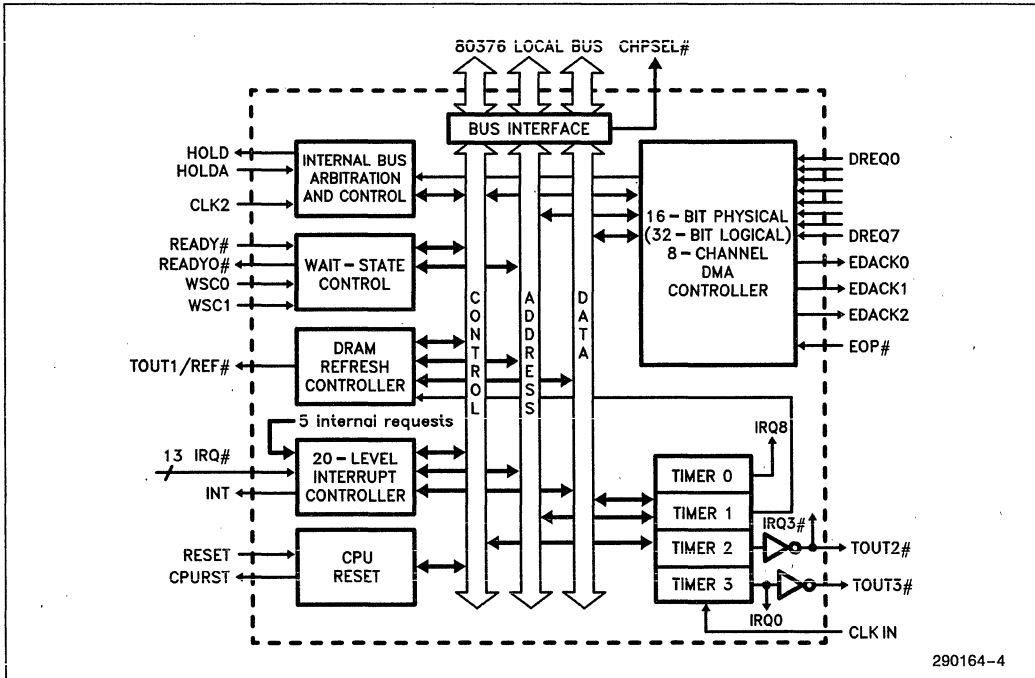


Figure 1-1. Architecture of the 82370

1.1.1 DMA CONTROLLER

The 82370 contains a high-performance, 8-channel DMA Controller. It provides a 32-bit internal data path. Through its 16-bit external physical data bus, it is capable of transferring data in any combination of bytes, words and double-words. The addresses of both source and destination can be independently incremented, decremented or held constant, and cover the entire 16-bit physical address space of the 80376. It can disassemble and assemble non-aligned data via a 32-bit internal temporary data storage register. Data transferred between devices of different data path widths can also be assembled and disassembled using the internal temporary data storage register. The DMA Controller can also transfer aligned data between I/O and memory on the fly, allowing data transfer rates up to 16 megabytes per second for an 82370 operating at 16 MHz. Figure 1-2 illustrates the functional components of the DMA Controller.

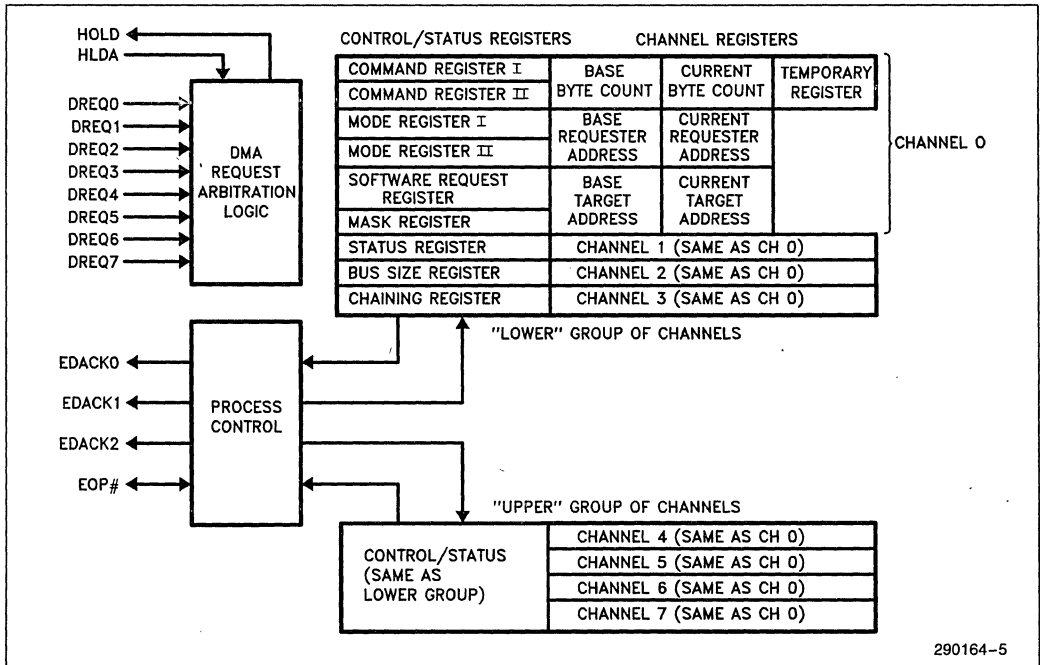
There are twenty-four general status and command registers in the 82370 DMA Controller. Through these registers any of the channels may be programmed into any of the possible modes. The operating modes of any one channel are independent of the operation of the other channels.

Each channel has three programmable registers which determine the location and amount of data to be transferred:

- Byte Count Register—Number of bytes to transfer. (24-bits)
- Requester Register — Byte Address of memory or peripheral which is requesting DMA service. (24-bits)
- Target Register — Byte Address of peripheral or memory which will be accessed. (24-bits)

There are also port addresses which, when accessed, cause the 82370 to perform specific functions. The actual data written doesn't matter, the act of writing to the specific address causes the command to be executed. The commands which operate in this mode are: Master Clear, Clear Terminal Count Interrupt Request, Clear Mask Register, and Clear Byte Pointer Flip-Flop.

DMA transfers can be done between all combinations of memory and I/O; memory-to-memory, memory-to-I/O, I/O-to-memory, and I/O-to-I/O. DMA service can be requested through software and/or hardware. Hardware DMA acknowledge signals are available for all channels (except channel 4) through an encoded 3-bit DMA acknowledge bus (EDACK0-2).



290164-5

Figure 1-2. 82370 DMA Controller

The 82370 DMA Controller transfers blocks of data (buffers) in three modes: Single Buffer, Buffer Auto-Initialize, and Buffer Chaining. In the Single Buffer Process, the 82370 DMA Controller is programmed to transfer one particular block of data. Successive transfers then require reprogramming of the DMA channel. Single Buffer transfers are useful in systems where it is known at the time the transfer begins what quantity of data is to be transferred, and there is a contiguous block of data area available.

The Buffer Auto-Initialize Process allows the same data area to be used for successive DMA transfers without having to reprogram the channel.

The Buffer Chaining Process allows a program to specify a list of buffer transfers to be executed. The 82370 DMA Controller, through interrupt routines, is reprogrammed from the list. The channel is reprogrammed for a new buffer before the current buffer transfer is complete. This pipelining of the channel programming process allows the system to allocate non-contiguous blocks of data storage space, and transfer all of the data with one DMA process. The buffers that make up the chain do not have to be in contiguous locations.

Channel priority can be fixed or rotating. Fixed priority allows the programmer to define the priority of DMA channels based on hardware or other fixed pa-

rameters. Rotating priority is used to provide peripherals access to the bus on a shared basis.

With fixed priority, the programmer can set any channel to have the current lowest priority. This allows the user to reset or manually rotate the priority schedule without reprogramming the command registers.

1.1.2 PROGRAMMABLE INTERVAL TIMERS

Four 16-bit programmable interval timers reside within the 82370. These timers are identical in function to the timers in the 82C54 Programmable Interval Timer. All four of the timers share a common clock input which can be independent of the system clock. The timers are capable of operating in six different modes. In all of the modes, the current count can be latched and read by the 80376 at any time, making these very versatile event timers. Figure 1-3 shows the functional components of the Programmable Interval Timers.

The outputs of the timers are directed to key system functions, making system design simpler. Timer 0 is routed directly to an interrupt input and is not available externally. This timer would typically be used to generate time-keeping interrupts.

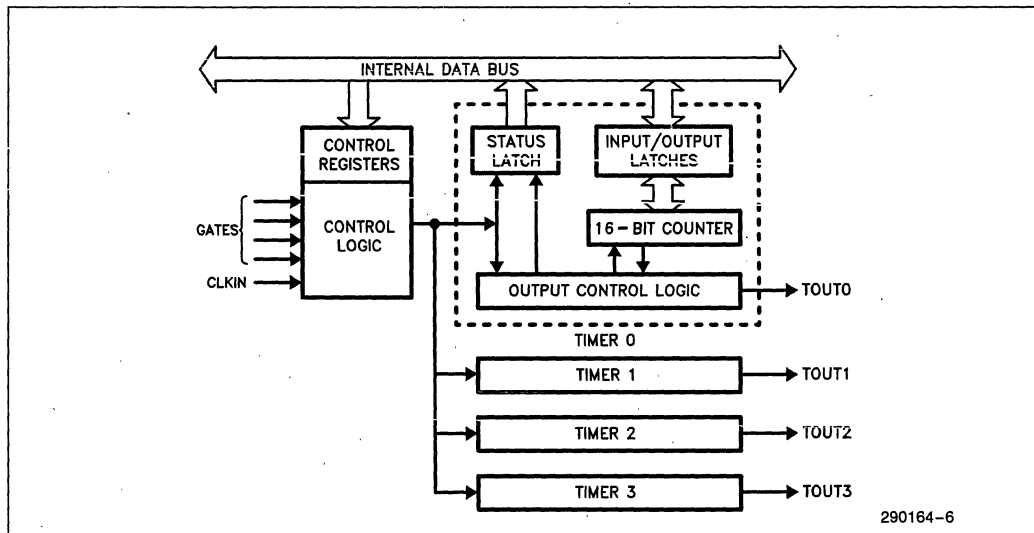


Figure 1-3. Programmable Interval Timers—Block Diagram

290164-6

Timers 1 and 2 have outputs which are available for general timer/counter purposes as well as special functions. Timer 1 is routed to the refresh control logic to provide refresh timing. Timer 2 is connected to an interrupt request input to provide other timer functions. Timer 3 is a general purpose timer/counter whose output is available to external hardware. It is also connected internally to the interrupt request which defaults to the highest priority (IRQ0).

1.1.3 INTERRUPT CONTROLLER

The 82370 has the equivalent of three enhanced 82C59A Programmable Interrupt Controllers. These controllers can all be operated in the Master Mode, but the priority is always as if they were cascaded. There are 15 interrupt request inputs provided for the user, all of which can be inputs from external slave interrupt controllers. Cascading 82C59As to these request inputs allows a possible total of 120 external interrupt requests. Figure 1-4 is a block diagram of the 82370 Interrupt Controller.

Each of the interrupt request inputs can be individually programmed with its own interrupt vector, allowing more flexibility in interrupt vector mapping than

was available with the 82C59A. An interrupt is provided to alert the system that an attempt is being made to program the vectors in the method of the 82C59A. This provides compatibility of existing software that used the 82C59A or 8259A with new designs using the 82370.

In the event of an unrequested or otherwise erroneous interrupt acknowledge cycle, the 82370 Interrupt Controller issues a default vector. This vector, programmed by the system software, will alert the system of unsolicited interrupts of the 80376.

The functions of the 82370 Interrupt Controller are identical to the 82C59A, except in regards to programming the interrupt vectors as mentioned above. Interrupt request inputs are programmable as either edge or level triggered and are software maskable. Priority can be either fixed or rotating and interrupt requests can be nested.

Enhancements are added to the 82370 for cascading external interrupt controllers. Master to Slave handshaking takes place on the data bus, instead of dedicated cascade lines.

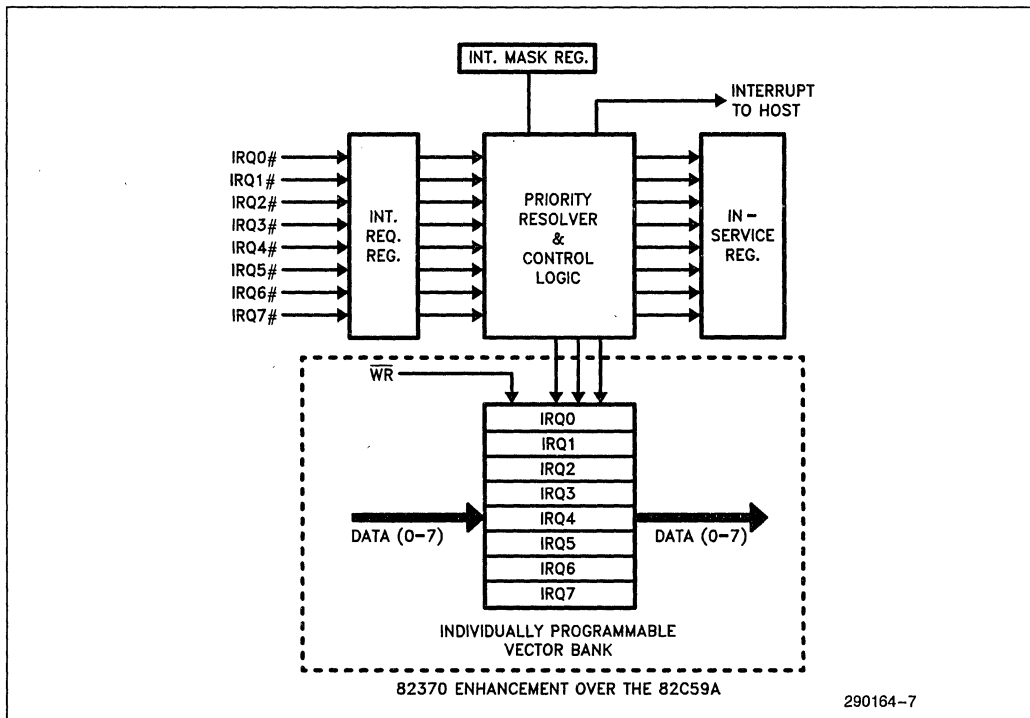


Figure 1-4. 82370 Interrupt Controller—Block Diagram

1.1.4 WAIT STATE GENERATOR

The Wait State Generator is a programmable READY generation circuit for the 80376 bus. A peripheral requiring wait states can request the Wait State Generator to hold the processor's READY input inactive for a predetermined number of bus states. Six different wait state counts can be programmed into the Wait State Generator by software; three for memory accesses and three for I/O accesses. A block diagram of the 82370 Wait State Generator is shown in Figure 1-5.

The peripheral being accessed selects the required wait state count by placing a code on a 2-bit wait state select bus. This code along with the M/IO# signal from the bus master is used to select one of six internal 4-bit wait state registers which has been programmed with the desired number of wait states. From zero to fifteen wait states can be programmed into the wait state registers. The Wait State generator tracks the state of the processor or current bus master at all times, regardless of which device is the current bus master and regardless of whether or not the wait state generator is currently active.

The 82370 Wait State Generator is disabled by making the select inputs both high. This allows hardware which is intelligent enough to generate its own ready signal to be accessed without penalty. As previously mentioned, deselecting the Wait State Generator does not disable its ability to determine the proper number of wait states due to pipeline status in subsequent bus cycles.

The number of wait states inserted into a pipelined bus cycle is the value in the selected wait state register. If the bus master is operating in the non-pipelined mode, the Wait State Generator will increase the number of wait states inserted into the bus cycle by one.

On reset, the Wait State Generator's registers are loaded with the value FFH, giving the maximum number of wait states for any access in which the wait state select inputs are active.

1.1.5 DRAM REFRESH CONTROLLER

The 82370 DRAM Refresh Controller consists of a 24-bit refresh address counter and bus arbitration logic. The output of Timer 1 is used to periodically request a refresh cycle. When the controller receives the request, it requests access to the system bus through the HOLD signal. When bus control is acknowledged by the processor or current bus master, the refresh controller executes a memory read operation at the address currently in the Refresh Address Register. At the same time, it activates a refresh signal (REF#) that the memory uses to force a refresh instead of a normal read. Control of the bus is transferred to the processor at the completion of this cycle. Typically a refresh cycle will take six clock cycles to execute on an 80376 bus.

The 82370 DRAM Refresh Controller has the highest priority when requesting bus access and will interrupt any active DMA process. This allows large blocks of data to be moved by the DMA controller without affecting the refresh function. Also the DMA controller is not required to completely relinquish the bus, the refresh controller simply steals a bus cycle between DMA accesses.

The amount by which the refresh address is incremented is programmable to allow for different bus widths and memory bank arrangements.

1.1.6 CPU RESET FUNCTION

The 82370 contains a special reset function which can respond to hardware reset signals as well as a

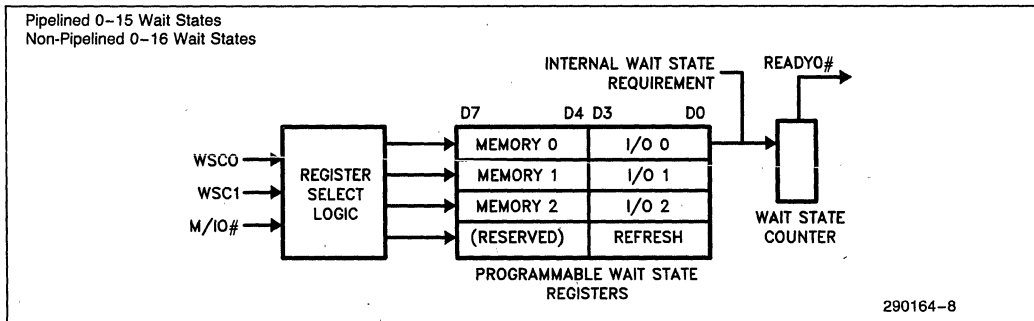


Figure 1-5. 82370 Wait State Generator—Block Diagram

software reset command. The circuit will hold the 80376's RESET line active while an external hardware reset signal is present at its RESET input. It can also reset the 80376 processor as the result of a software command. The software reset command causes the 82370 to hold the processor's RESET line active for a minimum of 62 clock cycles. The 80376 requires that its RESET line be held active for a minimum of 80 clock cycles to re-initialize. For a more detailed explanation and solution, see Appendix D (System Notes).

The 82370 can be programmed to sense the shutdown detect code on the status lines from the 80376. If the Shutdown Detect function is enabled, the 82370 will automatically reset the processor. A diagnostic register is available which can be used to determine the cause of reset.

1.1.7 REGISTER MAP RELOCATION

After a hardware reset, the internal registers of the 82370 are located in I/O space beginning at port address 0000H. The map of the 82370's registers is relocatable via a software command. The default mapping places the 82370 between I/O addresses 0000H and 00DBH. The relocation register allows this map to be moved to any even 256-byte boundary in the processor's 16-bit I/O address space or any even 64 kbyte boundary in the 24-bit memory address space.

1.2 Host Interface

The 82370 is designed to operate efficiently on the local bus of an 80376 microprocessor. The control signals of the 82370 are identical in function to those of the 80376. As a slave, the 82370 operates with all of the features available on the 80376 bus. When the 82370 is in the Master Mode, it looks identical to an 80376 to the connected devices.

The 82370 monitors the bus at all times, and determines whether the current bus cycle is a pipelined or non-pipelined access. All of the status signals of the processor are monitored.

The control, status, and data registers within the 82370 are located at fixed addresses relative to each other, but the group can be relocated to either memory or I/O space and to different locations within those spaces.

As a Slave device, the 82370 monitors the control/status lines of the CPU. The 82370 will generate all of the wait states it needs whenever it is accessed. This allows the programmer the freedom of access-

ing 82370 registers without having to insert NOPs in the program to wait for slower 82370 internal registers.

The 82370 can determine if a current bus cycle is a pipelined or a non-pipelined cycle. It does this by monitoring the ADS#, NA# and READY# signals and thereby keeping track of the current state of the 80376.

As a bus master, the 82370 looks like an 80376 to the rest of the system. This enables the designer greater flexibility in systems which include the 82370. The designer does not have to alter the interfaces of any peripherals designed to operate with the 80376 to accommodate the 82370. The 82370 will access any peripherals on the bus in the same manner as the 80376, including recognizing pipelined bus cycles.

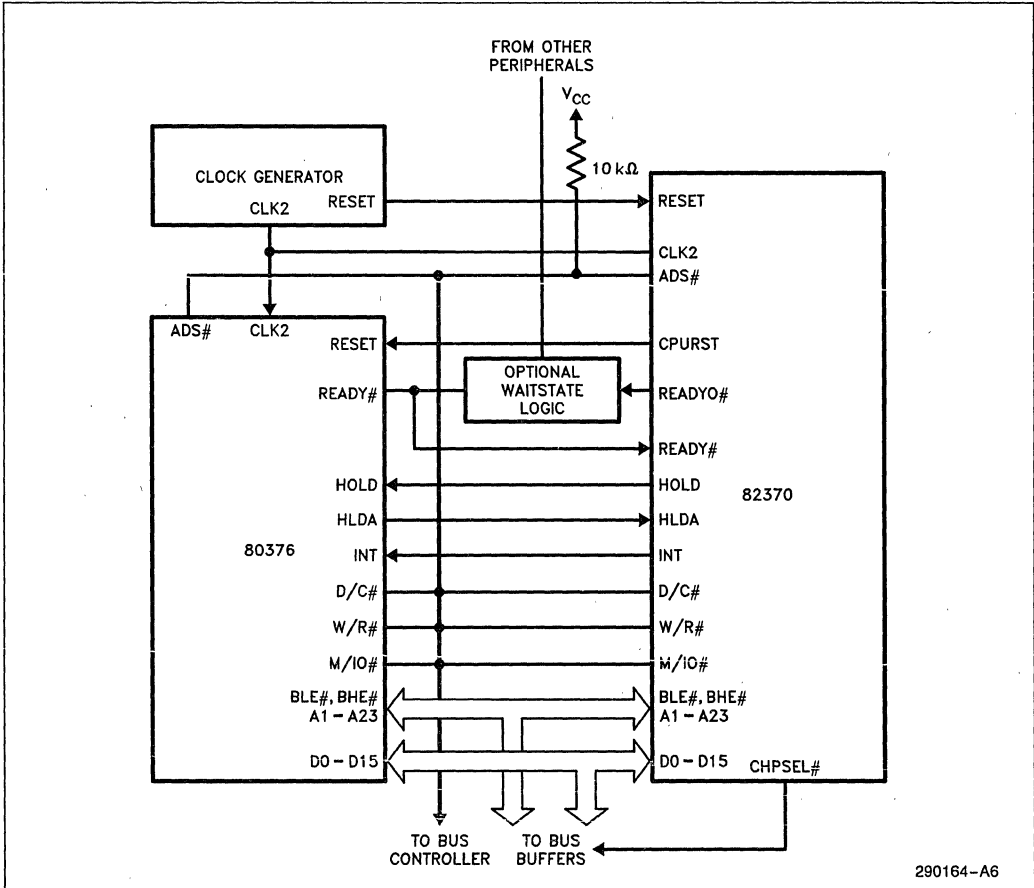
The 82370 is accessed as an 8-bit peripheral. The 80376 places the data of all 8-bit accesses either on D(0-7) or D(8-15). The 82370 will only accept data on these lines when in the Slave Mode. When in the Master Mode, the 82370 is a full 16-bit machine, sending and receiving data in the same manner as the 80376.

2.0 80376 HOST INTERFACE

The 82370 contains a set of interface signals to operate efficiently with the 80376 host processor. These signals were designed so that minimal hardware is needed to connect the 82370 to the 80376. Figure 2-1 depicts a typical system configuration with the 80376 processor. As shown in the diagram, the 82370 is designed to interface directly with the 80376 bus.

Since the 82370 resides on the opposite side of the data bus transceivers with respect to the rest of the system peripherals, it is important to note that the transceivers should be controlled so that contention between the data bus transceivers and the 82370 will not occur. In order to ease the implementation of this, the 82370 activates the CHPSEL# signal which indicates that the 82370 has been addressed and may output data. This signal should be included in the direction and enable control logic of the transceiver. When any of the 82370 internal registers are read, the data bus transceivers should be disabled so that only the 82370 will drive the local bus.

This section describes the basic bus functions of the 82370 to show how this device interacts with the 80376 processor. Other signals which are not directly related to the host interface will be discussed in their associated functional block description.



290164-A6

Figure 2-1. 80376/82370 System Configuration

2.1 Master and Slave Modes

At any time, the 82370 acts as either a Slave device or a Master device in the system. Upon reset, the 82370 will be in the Slave Mode. In this mode, the 80376 processor can read/write into the 82370 internal registers. Initialization information may be programmed into the 82370 during Slave Mode.

When DMA service (including DRAM Refresh Cycles generated by the 82370) is requested, the 82370 will request and subsequently get control of the 80376 local bus. This is done through the HOLD and HLDA (Hold Acknowledge) signals. When the 80376 proc-

essor responds by asserting the HLDA signal, the 82370 will switch into Master Mode and perform DMA transfers. In this mode, the 82370 is the bus master of the system. It can read/write data from/to memory and peripheral devices. The 82370 will return to the Slave Mode upon completion of DMA transfers, or when HLDA is negated.

2.2 80376 Interface Signals

As mentioned in the Architecture section, the Bus Interface module of the 82370 (see Figure 1-1) contains signals that are directly connected to the 80376 host processor. This module has separate

16-bit Data and 24-bit Address busses. Also, it has additional control signals to support different bus operations on the system. By residing on the 80376 local bus, the 82370 shares the same address, data and control lines with the processor. The following subsections discuss the signals which interface to the 80376 host processor.

2.2.1 CLOCK (CLK2)

The CLK2 input provides fundamental timing for the 82370. It is divided by two internally to generate the 82370 internal clock. Therefore, CLK2 should be driven with twice the 80376's frequency. In order to maintain synchronization with the 80376 host processor, the 82370 and the 80376 should share a common clock source.

The internal clock consists of two phases: PHI1 and PHI2. Each CLK2 period is a phase of the internal clock. PHI2 is usually used to sample input and set up internal signals and PHI1 is for latching internal data. Figure 2-2 illustrates the relationship of CLK2 and the 82370 internal clock signals. The CPURST signal generated by the 82370 guarantees that the 80376 will wake up in phase with PHI1.

2.2.2 DATA BUS (D₀-D₁₅)

This 16-bit three-state bidirectional bus provides a general purpose data path between the 82370 and the system. These pins are tied directly to the corresponding Data Bus pins of the 80376 local bus. The Data Bus is also used for interrupt vectors generated by the 82370 in the Interrupt Acknowledge cycle.

During Slave I/O operations, the 82370 expects a single byte to be written or read. When the 80376 host processor writes into the 82370, either D₀-D₇ or D₈-D₁₅ will be latched into the 82370, depending

upon whether Byte Enable bit BLE# is 0 or 1 (see Table 2-1). When the 80376 host processor reads from the 82370, the single byte data will be duplicated twice on the Data Bus; i.e. on D₀-D₇ and D₈-D₁₅.

During Master Mode, the 82370 can transfer 16-, and 8-bit data between memory (or I/O devices) and I/O devices (or memory) via the Data Bus.

2.2.3 ADDRESS BUS (A₂₃-A₁)

These three-state bidirectional signals are connected directly to the 80376 Address Bus. In the Slave Mode, they are used as input signals so that the processor can address the 82370 internal ports/registers. In the Master Mode, they are used as output signals by the 82370 to address memory and peripheral devices. The Address Bus is capable of addressing 16 Mbytes of physical memory space (000000H to FFFFFFFH), and 64 Kbytes of I/O addresses.

2.2.4 BYTE ENABLE (BHE#, BLE#)

The Byte Enable pins BHE# and BLE# select the specific byte(s) in the word addressed by A₁-A₂₃. During Master Mode operation, it is used as an output by the 82370 to address memory and I/O locations. The definition of BHE# and BLE# is further illustrated in Table 2-1.

NOTE:

The 82370 will activate BHE# when output in Master Mode. For a more detailed explanation and its solutions, see Appendix D (System Notes).

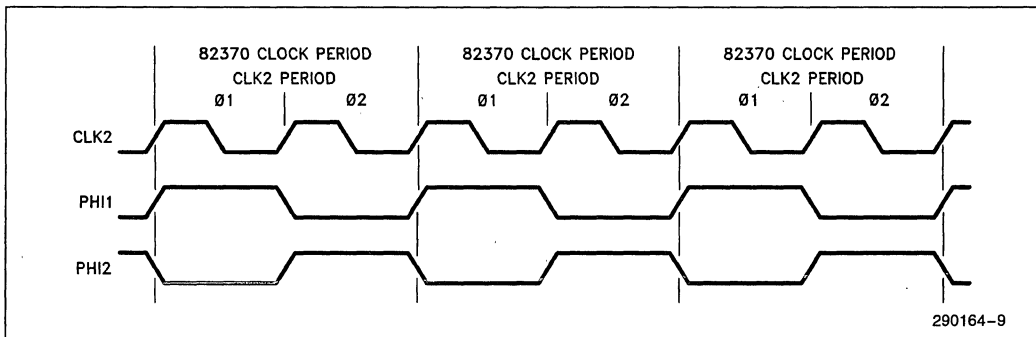


Figure 2-2. CLK2 and 82370 Internal Clock

As an output (Master Mode):

Table 2-1. Byte Enable Signals

BHE #	BLE #	Byte to be Accessed Relative to A ₂₃ -A ₁	Logical Byte Presented on Data Bus During WRITE Only*	
			D ₁₅ -D ₈	D ₇ -D ₀
0	0	0, 1	B	A
0	1	1	A	A
1	0	0	U	A
1	1	(Not Used)		

U = Undefined
 A = Logical D₀-D₇
 B = Logical D₈-D₁₅

***NOTE:**

Actual number of bytes accessed depends upon the programmed data path width.

Table 2-2. Bus Cycle Definition

M/IO #	D/C #	W/R #	As INPUTS	As OUTPUTS
0	0	0	Interrupt Acknowledge	NOT GENERATED
0	0	1	UNDEFINED	NOT GENERATED
0	1	0	I/O Read	I/O Read
0	1	1	I/O Write	I/O Write
1	0	0	UNDEFINED	NOT GENERATED
1	0	1	HALT if A ₁ = 1 SHUTDOWN if A ₁ = 0	NOT GENERATED
1	1	0	Memory Read	Memory Read
1	1	1	Memory Write	Memory Write

**2.2.5 BUS CYCLE DEFINITION SIGNALS
(D/C#, W/R#, M/IO#)**

These three-state bidirectional signals define the type of bus cycle being performed. W/R# distinguishes between write and read cycles. D/C# distinguishes between processor data and control cycles. M/IO# distinguishes between memory and I/O cycles.

During Slave Mode, these signals are driven by the 80376 host processor; during Master Mode, they are driven by the 82370. In either mode, these signals will be valid when the Address Status (ADS#) is driven LOW. Exact bus cycle definitions are given in Table 2-2. Note that some combinations are recognized as inputs, but not generated as outputs. In the Master Mode, D/C# is always HIGH.

2.2.6 ADDRESS STATUS (ADS#)

This signal indicates that a valid address (A₁-A₂₃, BHE#, BLE#) and bus cycle definition (W/R#, D/C#, M/IO#) is being driven on the bus. In the Master Mode, it is driven by the 82370 as an output. In the Slave Mode, this signal is monitored as

an input by the 82370. By the current and past status of ADS# and the READY# input, the 82370 is able to determine, during Slave Mode, if the next bus cycle is a pipelined address cycle. ADS# is asserted during T1 and T2P bus states (see Bus State Definition).

NOTE:

ADS# must be qualified with the rising edge of CLK2.

2.2.7 TRANSFER ACKNOWLEDGE (READY#)

This input indicates that the current bus cycle is complete. In the Master Mode, assertion of this signal indicates the end of a DMA bus cycle. In the Slave Mode, the 82370 monitors this input and ADS# to detect a pipelined address cycle. This signal should be tied directly to the READY# input of the 80376 host processor.

2.2.8 NEXT ADDRESS REQUEST (NA#)

This input is used to indicate to the 82370 in the Master Mode that the system is requesting address

pipelining. When driven LOW by either memory or peripheral devices during Master Mode, it indicates that the system is prepared to accept a new address and bus cycle definition signals from the 82370 before the end of the current bus cycle. If this input is active when sampled by the 82370, the next address is driven onto the bus, provided a bus request is already pending internally.

This input pin is monitored only in the Master Mode. In the Slave Mode, the 82370 uses the ADS# and READY# signals to determine address pipelining cycles, and NA# will be ignored.

2.2.9 RESET (RESET, CPURST)

RESET

This synchronous input suspends any operation in progress and places the 82370 in a known initial state. Upon reset, the 82370 will be in the Slave Mode waiting to be initialized by the 80376 host processor. The 82370 is reset by asserting RESET for 15 or more CLK2 periods. When RESET is asserted, all other input pins are ignored, and all other bus pins are driven to an idle bus state as shown in Table 2-3. The 82370 will determine the phase of its internal clock following RESET going inactive.

RESET is level-sensitive and must be synchronous to the CLK2 signal. The RESET setup and hold time requirements are shown in Figure 2-3.

Table 2-3. Output Signals Following RESET

Signal	Level
A ₁ -A ₂₃ , D ₀ -D ₁₅ , BHE#, BLE#	Float
D/C#, W/R#, M/IO#, ADS#	Float
READYO#	'1'
EOP#	'1' (Weak Pull-UP)
EDACK2-EDACK0	'100'
HOLD	'0'
INT	UNDEFINED*
TOUT1/REF#, TOUT2#/IRQ3#, TOUT3#	UNDEFINED*
CPURST	'0'
CHPSEL#	'1'

***NOTE:**
The Interrupt Controller and Programmable Interval Timer are initialized by software commands.

CPURST

This output signal is used to reset the 80376 host processor. It will go active (HIGH) whenever one of the following events occurs: a) 82370's RESET input is active; b) a software RESET command is issued to the 82370; or c) when the 82370 detects a processor Shutdown cycle and when this detection feature is enabled (see CPU Reset and Shutdown Detect). When activated, CPURST will be held active for 62 clocks. The timing of CPURST is such that the 80376 processor will be in synchronization with the 82370. This timing is shown in Figure 2-4.

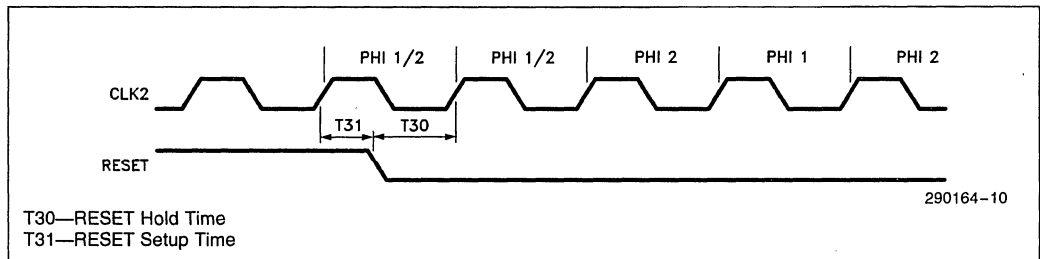


Figure 2-3. RESET Timing

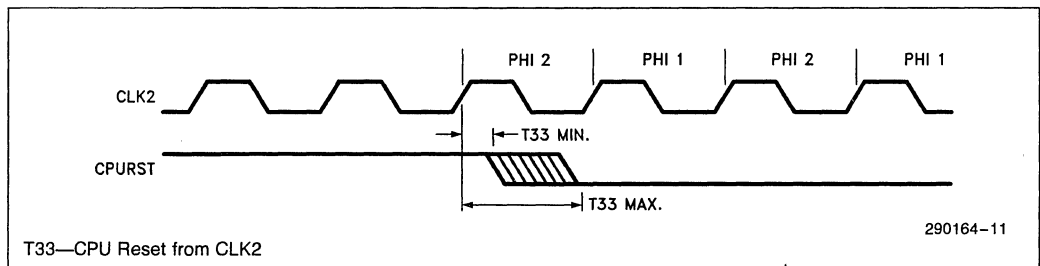


Figure 2-4. CPURST Timing

2.2.10 INTERRUPT OUT (INT)

This output pin is used to signal the 80376 host processor that one or more interrupt requests (either internal or external) are pending. The processor is expected to respond with an Interrupt Acknowledge cycle. This signal should be connected directly to the Maskable Interrupt Request (INTR) input of the 80376 host processor.

2.3 82370 Bus Timing

The 82370 internally divides the CLK2 signal by two to generate its internal clock. Figure 2-2 showed the relationship of CLK2 and the internal clock which consists of two phases: PHI1 and PHI2. Each CLK2 period is a phase of the internal clock.

In the 82370, whether it is in the Master or Slave Mode, the shortest time unit of bus activity is a bus state. A bus state, which is also referred as a 'T-state', is defined as one 82370 PHI2 clock period (i.e. two CLK2 periods). Recall in Table 2-2 various types of bus cycles in the 82370 are defined by the M/I/O#, D/C# and W/R# signals. Each of these bus cycles is composed of two or more bus states. The length of a bus cycle depends on when the READY# input is asserted (i.e. driven LOW).

2.3.1 ADDRESS PIPELINING

The 82370 supports Address Pipelining as an option in both the Master and Slave Mode. This feature typically allows a memory or peripheral device to operate with one less wait state than would otherwise be required. This is possible because during a pipelined cycle, the address and bus cycle definition of the next cycle will be generated by the bus master while waiting for the end of the current cycle to be acknowledged. The pipelined bus is especially well suited for an interleaved memory environment. For 16 MHz interleaved memory designs with 100 ns access time DRAMs, zero wait state memory accesses can be achieved when pipelined addressing is selected.

In the Master Mode, the 82370 is capable of initiating, on a cycle-by-cycle basis, either a pipelined or non-pipelined access depending upon the state of the NA# input. If a pipelined cycle is requested (indicated by NA# being driven LOW), the 82370 will drive the address and bus cycle definition of the next cycle as soon as there is an internal bus request pending.

In the Slave Mode, the 82370 is constantly monitoring the ADS# and READY# signals on the processor local bus to determine if the current bus cycle is

a pipelined cycle. If a pipelined cycle is detected, the 82370 will request one less wait state from the processor if the Wait State Generator feature is selected. On the other hand, during an 82370 internal register access in a pipelined cycle, it will make use of the advance address and bus cycle information. In all cases, Address Pipelining will result in a savings of one wait state.

2.3.2 MASTER MODE BUS TIMING

When the 82370 is in the Master Mode, it will be in one of six bus states. Figure 2-5 shows the complete bus state diagram of the Master Mode, including pipelined address states. As seen in the figure, the 82370 state diagram is very similar to that of the 80376. The major difference is that in the 82370, there is no Hold state. Also, in the 82370, the conditions for some state transitions depend upon whether it is the end of a DMA process.

NOTE:

The term 'end of a DMA process' is loosely defined here. It depends on the DMA modes of operation as well as the state of the EOP# and DREQ inputs. This is explained in detail in section 3—DMA Controller.

The 82370 will enter the idle state, T_i, upon RESET and whenever the internal address is not available at the end of a DMA cycle or at the end of a DMA process. When address pipelining is not used (NA# is not asserted), a new bus cycle always begins with state T₁. During T₁, address and bus cycle definition signals will be driven on the bus. T₁ is always followed by T₂.

if a bus cycle is not acknowledged (with READY#) during T₂ and NA# is negated, T₂ will be repeated. When the end of the bus cycle is acknowledged during T₂, the following state will be T₁ of the next bus cycle (if the internal address latch is loaded and if this is not the end of the DMA process). Otherwise, the T_i state will be entered. Therefore, if the memory or peripheral accessed is fast enough to respond within the first T₂, the fastest non-pipelined cycle will take one T₁ and one T₂ state.

Use of the address pipelining feature allows the 82370 to enter three additional bus states: T_{1P}, T_{2P} and T_{2i}. T_{1P} is the first bus state of a pipelined bus cycle. T_{2P} follows T_{1P} (or T₂) if NA# is asserted when sampled. The 82370 will drive the bus with the address and bus cycle definition signals of the next cycle during T_{2P}. From the state diagram, it can be seen that after an idle state T_i, the first bus cycle must begin with T₁, and is therefore a non-pipelined bus cycle. The next bus cycle can be pipelined if

NA# is asserted and the previous bus cycle ended in a T2P state. Once the 82370 is in a pipelined cycle and provided that NA# is asserted in subsequent cycles, the 82370 will be switching between T1P and T2P states. If the end of the current bus cycle is not acknowledged by the READY# input, the 82370 will extend the cycle by adding T2P states. The fastest pipelined cycle will consist of one T1P and one T2P state.

The 82370 will enter state T2i when NA# is asserted and when one of the following two conditions occurs. The first condition is when the 82370 is in state T2. T2i will be entered if READY# is not asserted and there is no next address available. This situation is similar to a wait state. The 82370 will stay in T2i for as long as this condition exists. The second condition which will cause the 82370 to enter T2i is when the 82370 is in state T1P. Before going to state T2P, the 82370 needs to wait in state T2i until the next address is available. Also, in both cases, if the DMA process is complete, the 82370 will enter the T2i state in order to finish the current DMA cycle.

Figure 2-6 is a timing diagram showing non-pipelined bus accesses in the Master Mode. Figure 2-7 shows the timing of pipelined accesses in the Master Mode.

2.3.3 SLAVE MODE BUS TIMING

Figure 2-8 shows the Slave Mode bus timing in both pipelined and non-pipelined cycles when the 82370 is being accessed. Recall that during Slave Mode, the 82370 will constantly monitor the ADS# and READY# signals to determine if the next cycle is pipelined. In Figure 2-8, the first cycle is non-pipelined and the second cycle is pipelined. In the pipelined cycle, the 82370 will start decoding the address and bus cycle signals one bus state earlier than in a non-pipelined cycle.

The READY# input signal is sampled by the 80376 host processor to determine the completion of a bus cycle. This occurs during the end of every T2, T2i and T2P state. Normally, the output of the 82370 Wait State Generator, READYO#, is directly connected to the READY# input of the 80376 host processor and the 82370. In such case, READYO# and READY# will be identical (see Wait State Generator).

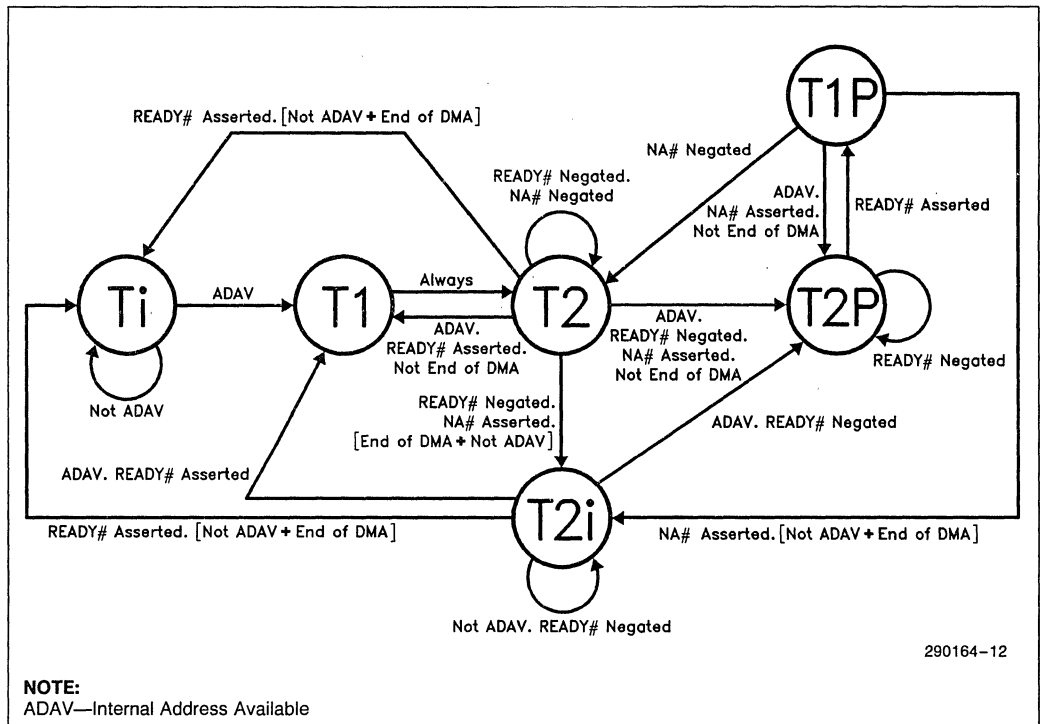


Figure 2-5. Master Mode State Diagram

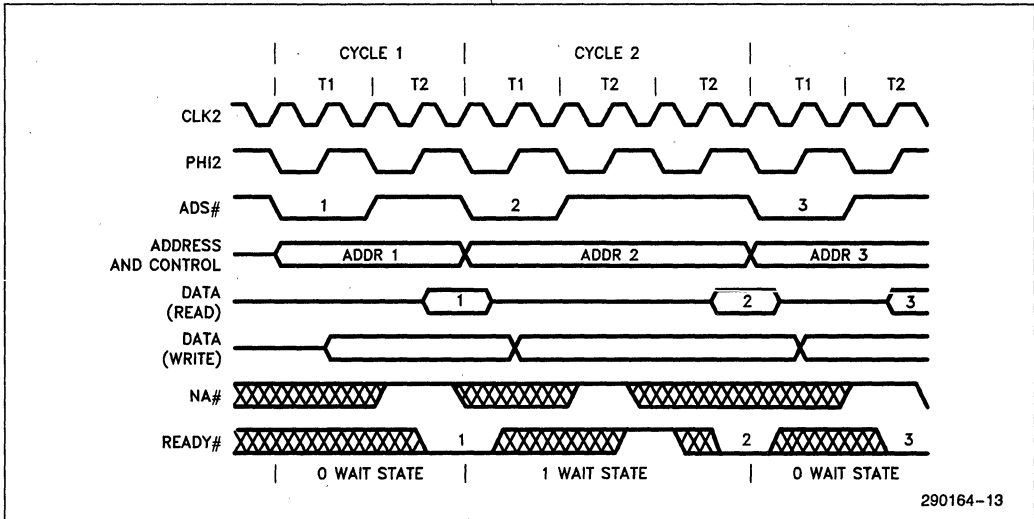


Figure 2-6. Non-Pipelined Bus Cycles

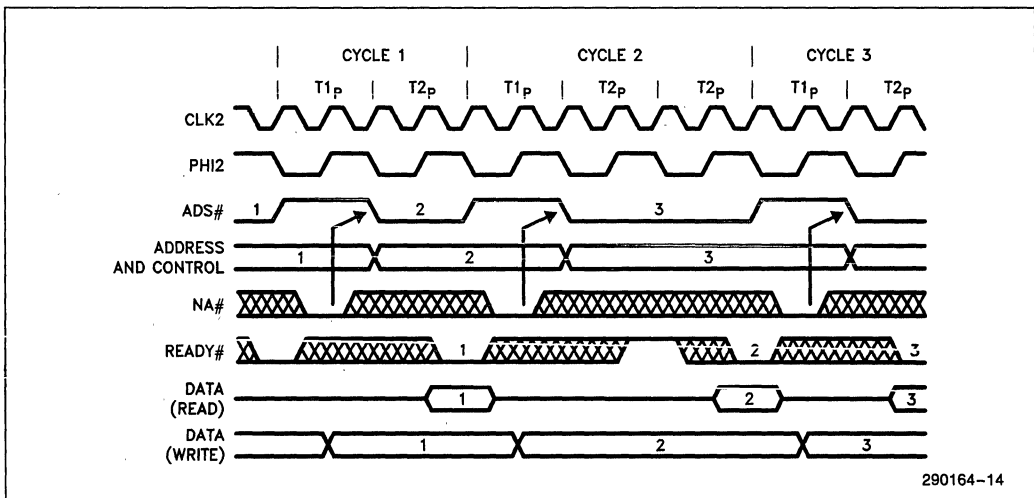


Figure 2-7. Pipelined Bus Cycles

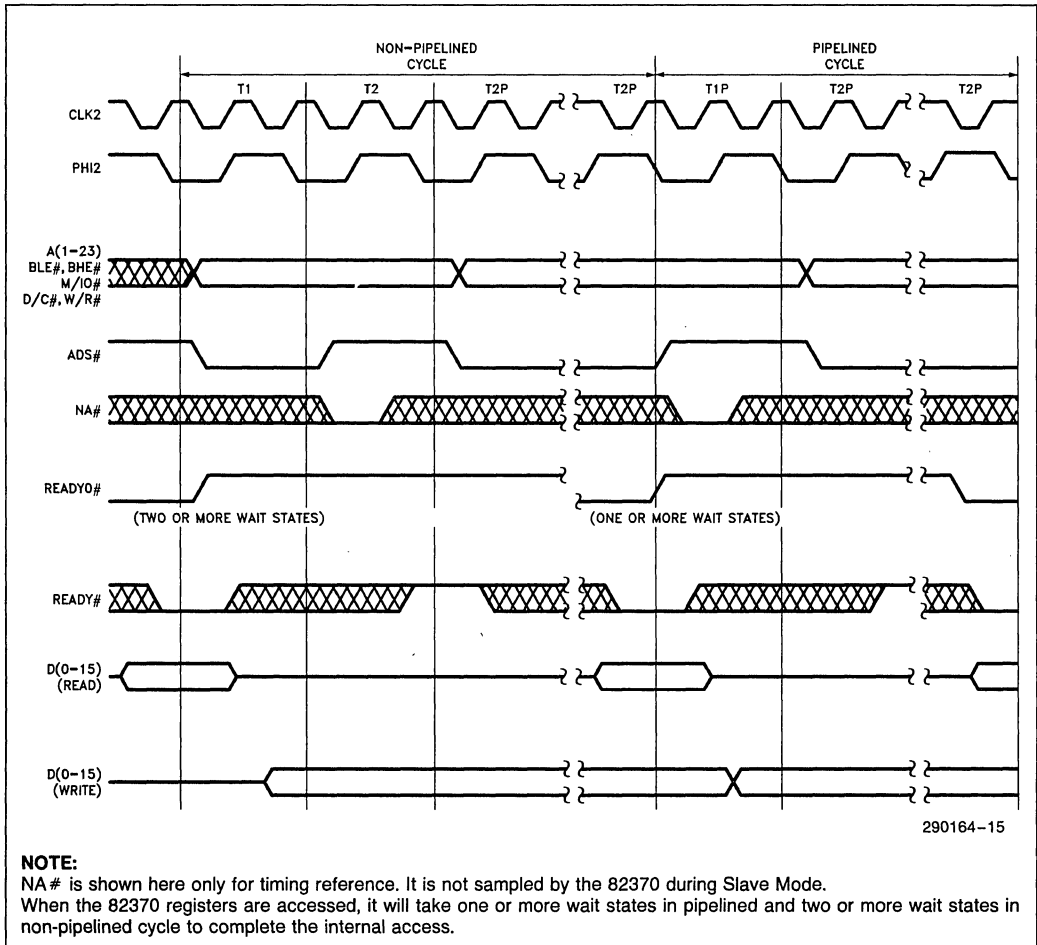


Figure 2-8. Slave Read/Write Timing

3.0 DMA CONTROLLER

The 82370 DMA Controller is capable of transferring data between any combination of memory and/or I/O, with any combination of data path widths. The 82370 DMA Controller can be programmed to accommodate 8- or 16-bit devices. With its 16-bit external data path, it can transfer data in units of byte or a word. Bus bandwidth is optimized through the use of an internal temporary register which can disassemble or assemble data to or from either an aligned or non-aligned destination or source. Figure 3-1 is a block diagram of the 82370 DMA Controller.

The 82370 has eight channels of DMA. Each channel operates independently of the others. Within the operation of the individual channels, there are many different modes of data transfer available. Many of the operating modes can be intermixed to provide a very versatile DMA controller.

3.1 Functional Description

In describing the operation of the 82370's DMA Controller, close attention to terminology is required. Be-

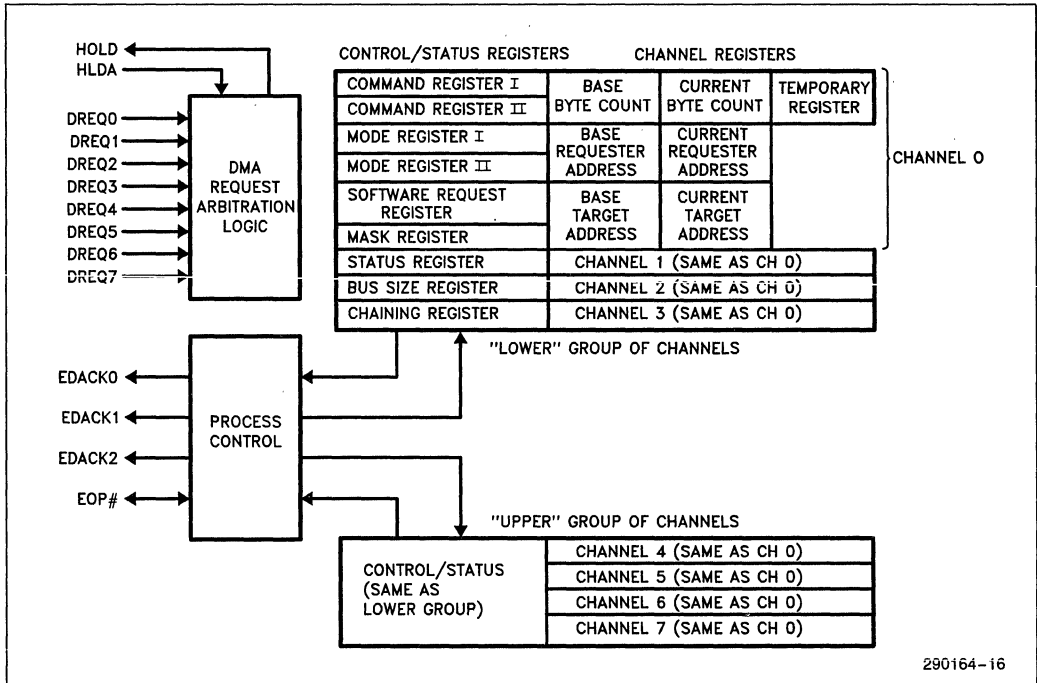


Figure 3-1. 82370 DMA Controller Block Diagram

fore entering the discussion of the function of the 82370 DMA Controller, the following explanations of some of the terminology used herein may be of benefit. First, a few terms for clarification:

DMA PROCESS—A DMA process is the execution of a programmed DMA task from beginning to end. Each DMA process requires initial programming by the host 80376 microprocessor.

BUFFER—A contiguous block of data.

BUFFER TRANSFER—The action required by the DMA to transfer an entire buffer.

DATA TRANSFER—The DMA action in which a group of bytes or words are moved between devices by the DMA Controller. A data transfer operation may involve movement of one or many bytes.

BUS CYCLE—Access by the DMA to a single byte or word.

Each DMA channel consists of three major components. These components are identified by the contents of programmable registers which define the

memory or I/O devices being serviced by the DMA. They are the Target, the Requester, and the Byte Count. They will be defined generically here and in greater detail in the DMA register definition section.

The Requester is the device which requires service by the 82370 DMA Controller, and makes the request for service. All of the control signals which the DMA monitors or generates for specific channels are logically related to the Requester. Only the Requester is considered capable of initiating or terminating a DMA process.

The Target is the device with which the Requester wishes to communicate. As far as the DMA process is concerned, the Target is a slave which is incapable of control over the process.

The direction of data transfer can be either from Requester to Target or from Target to Requester; i.e. each can be either a source or a destination.

The Requester and Target may each be either I/O or memory. Each has an address associated with it that can be incremented, decremented, or held constant. The addresses are stored in the Requester

Address Registers and Target Address Registers, respectively. These registers have two parts: one which contains the current address being used in the DMA process (Current Address Register), and one which holds the programmed base address (Base Address Register). The contents of the Base Registers are never changed by the 82370 DMA Controller. The Current Registers are incremented or decremented according to the progress of the DMA process.

The Byte Count is the component of the DMA process which dictates the amount of data which must be transferred. Current and Base Byte Count Registers are provided. The Current Byte Count Register is decremented once for each byte transferred by the DMA process. When the register is decremented past zero, the Byte Count is considered 'expired' and the process is terminated or restarted, depending on the mode of operation of the channel. The point at which the Byte Count expires is called 'Terminal Count' and several status signals are dependent on this event.

Each channel of the 82370 DMA Controller also contains a 32-bit Temporary Register for use in assembling and disassembling non-aligned data. The operation of this register is transparent to the user, although the contents of it may affect the timing of some DMA handshake sequences. Since there is data storage available for each channel, the DMA Controller can be interrupted without loss of data.

To avoid unexpected results, care should be taken in programming the byte count correctly when assembling and disassembling non-aligned data. For example:

Words to Bytes:

Transferring two words to bytes, but setting the byte count to three, will result in three bytes transferred and the final byte flushed.

Bytes to Words:

Transferring six bytes to three words, but setting the byte count to five, will result in the sixth byte transferred being undefined.

The 82370 DMA Controller is a slave on the bus until a request for DMA service is received via either a software request command or a hardware request signal. The host processor may access any of the control/status or channel registers at any time the 82370 is a bus slave. Figure 3-2 shows the flow of operations that the DMA Controller performs.

At the time a DMA service request is received, the DMA Controller issues a bus hold request to the host processor. The 82370 becomes the bus master when the host relinquishes the bus by asserting a

hold acknowledge signal. The channel to be serviced will be the one with the highest priority at the time the DMA Controller becomes the bus master. The DMA Controller will remain in control of the bus until the hold acknowledge signal is removed, or until the current DMA transfer is complete.

While the 82370 DMA Controller has control of the bus, it will perform the required data transfer(s). The type of transfer, source and destination addresses, and amount of data to transfer are programmed in the control registers of the DMA channel which received the request for service.

At completion of the DMA process, the 82370 will remove the bus hold request. At this time the 82370 becomes a slave again, and the host returns to being a master. If there are other DMA channels with requests pending, the controller will again assert the hold request signal and restart the bus arbitration and switching process.

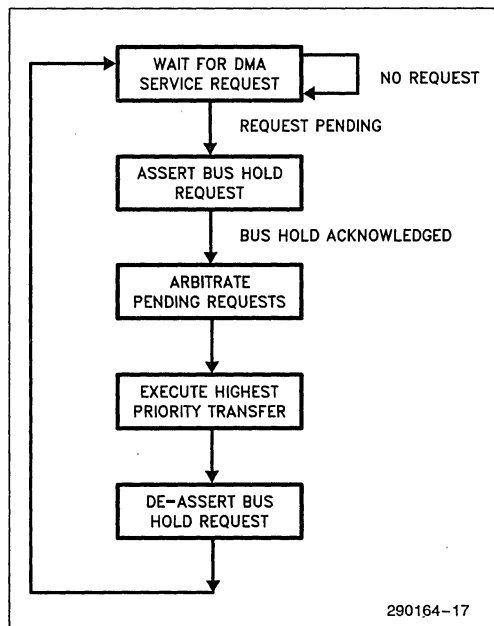


Figure 3-2. Flow of DMA Controller Operation

3.2 Interface Signals

There are fourteen control signals dedicated to the DMA process. They include eight DMA Channel Requests (DREQn), three Encoded DMA Acknowledge signals (EDACKn), Processor Hold and Hold Ac-

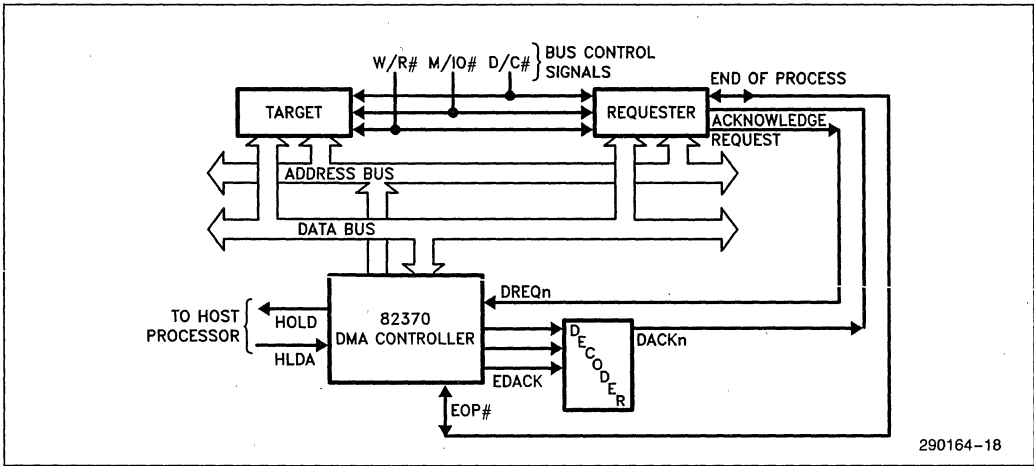


Figure 3-3. Requester, Target and DMA Controller Interconnection

290164-18

knowledge (HOLD, HLDA), and End-of-Process (EOP#). The DREQn inputs and EDACK (0-2) outputs are handshake signals to the devices requiring DMA service. The HOLD output and HLDA input are handshake signals to the host processor. Figure 3-3 shows these signals and how they interconnect between the 82370 DMA Controller, and the Requester and Target devices.

3.2.1 DREQn and EDACK (0-2)

These signals are the handshake signals between the peripheral and the 82370. When the peripheral requires DMA service, it asserts the DREQn signal of the channel which is programmed to perform the service. The 82370 arbitrates the DREQn against other pending requests and begins the DMA process after finishing other higher priority processes.

When the DMA service for the requested channel is in progress, the EDACK (0-2) signals represent the DMA channel which is accessing the Requester. The 3-bit code on the EDACK (0-2) lines indicates the number of the channel presently being serviced. Table 3-2 shows the encoding of these signals. Note that Channel 4 does not have a corresponding hardware acknowledge.

The DMA acknowledge (EDACK) signals indicate the active channel only during DMA accesses to the Requester. During accesses to the Target, EDACK (0-2) has the idle code (100). EDACK (0-2) can thus be used to select a Requester device during a transfer.

DREQn can be programmed as either an Asynchronous or Synchronous input. See section 3.4.1 for details on synchronous versus asynchronous operation of these pins.

Table 3-2. EDACK Encoding During a DMA Transfer

EDACK2	EDACK1	EDACK0	Active Channel
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	Target Access
1	0	1	5
1	1	0	6
1	1	1	7

The EDACKn signals are always active. They either indicate 'no acknowledge' or they indicate a bus access to the requester. The acknowledge code is either 100, for an idle DMA or during a DMA access to the Target, or 'n' during a Requester access, where n is the binary value representing the channel. A simple 3-line to 8-line decoder can be used to provide discrete acknowledge signals for the peripherals.

3.2.2 HOLD AND HLDA

The Hold Request (HOLD) and Hold Acknowledge (HLDA) signals are the handshake signals between the DMA Controller and the host processor. HOLD is an output from the 82370 and HLDA is an input. HOLD is asserted by the DMA Controller when there is a pending DMA request, thus requesting the processor to give up control of the bus so the DMA process can take place. The 80376 responds by asserting HLDA when it is ready to relinquish control of the bus.

The 82370 will begin operations on the bus one clock cycle after the HLDA signal goes active. For this reason, other devices on the bus should be in the slave mode when HLDA is active.

HOLD and HLDA should not be used to gate or select peripherals requesting DMA service. This is because of the use of DMA-like operations by the DRAM Refresh Controller. The Refresh Controller is arbitrated with the DMA Controller for control of the bus, and refresh cycles have the highest priority. A refresh cycle will take place between DMA cycles without relinquishing bus control. See section 3.4.3 for a more detailed discussion of the interaction between the DMA Controller and the DRAM Refresh Controller.

3.2.3 EOP#

EOP# is a bi-directional signal used to indicate the end of a DMA process. The 82370 activates this as an output during the T2 states of the last Requester bus cycle for which a channel is programmed to execute. The Requester should respond by either withdrawing its DMA request, or interrupting the host processor to indicate that the channel needs to be programmed with a new buffer. As an input, this signal is used to tell the DMA Controller that the peripheral being serviced does not require any more data to be transferred. This indicates that the current buffer is to be terminated.

EOP# can be programmed as either an Asynchronous or a Synchronous input. See section 3.4.1 for details on synchronous versus asynchronous operation of this pin.

3.3 Modes of Operation

The 82370 DMA Controller has many independent operating functions. When designing peripheral interfaces for the 82370 DMA Controller, all of the functions or modes must be considered. All of the channels are independent of each other (except in priority of operation) and can operate in any of the modes. Many of the operating modes, though independently programmable, affect the operation of other modes. Because of the large number of combinations possible, each programmable mode is discussed here with its affects on the operation of other modes. The entire list of possible combinations will not be presented.

Table 3-1 shows the categories of DMA features available in the 82370. Each of the five major categories is independent of the others. The sub-categories are the available modes within the major func-

Table 3-1. DMA Operating Modes

- I. TARGET/REQUESTER DEFINITION
 - a. Data Transfer Direction
 - b. Device Type
- II. BUFFER PROCESSES
 - a. Single Buffer Process
 - b. Buffer Auto-Initialize Process
 - c. Buffer Chaining Process
- III. DATA TRANSFER/HANDSHAKE MODES
 - a. Single Transfer Mode
 - b. Demand Transfer Mode
 - c. Block Transfer Mode
 - d. Cascade Mode
- IV. PRIORITY ARBITRATION
 - a. Fixed
 - b. Rotating
 - c. Programmable Fixed
- V. BUS OPERATION
 - a. Fly-By (Single-Cycle)/Two-Cycle
 - b. Data Path Width
 - c. Read, Write, or Verify Cycles

tion or mode category. The following sections explain each mode or function and its relation to other features.

3.3.1 TARGET/REQUESTER DEFINITION

All DMA transfers involve three devices: the DMA Controller, the Requester, and the Target. Since the devices to be accessed by the DMA Controller vary widely, the operating characteristics of the DMA Controller must be tailored to the Requester and Target devices.

The Requester can be defined as either the source or the destination of the data to be transferred. This is done by specifying a Write or a Read transfer, respectively. In a Read transfer, the Target is the data source and the Requester is the destination for the data. In a Write transfer, the Requester is the source and the Target is the destination.

The Requester and Target addresses can each be independently programmed to be incremented, decremented, or held constant. As an example, the 82370 is capable of reversing a string of data by having the Requester address increment and the Target address decrement in a memory-to-memory transfer.

3.3.2 BUFFER TRANSFER PROCESSES

The 82370 DMA Controller allows three programmable Buffer Transfer Processes. These processes define the logical way in which a buffer of data is accessed by the DMA.

The three Buffer Transfer Processes include the Single Buffer Process, the Buffer Auto-Initialize Process, and the Buffer Chaining Process. These processes require special programming considerations. See the DMA Programming section for more details on setting up the Buffer Transfer Processes.

Single Buffer Process

The Single Buffer Process allows the DMA channel to transfer only one buffer of data. When the buffer has been completely transferred (Current Byte Count decremented past zero or EOP# input active), the DMA process ends and the channel becomes idle. In order for that channel to be used again, it must be reprogrammed.

The Single Buffer Process is usually used when the amount of data to be transferred is known exactly, and it is also known that there is not likely to be any data to follow before the operating system can reprogram the channel.

Buffer Auto-Initialize Process

The Buffer Auto-Initialize Process allows multiple groups of data to be transferred to or from a single buffer. This process does not require reprogramming. The Current Registers are automatically reprogrammed from the Base Registers when the current process is terminated, either by an expired Byte Count or by an external EOP# signal. The data transferred will always be between the same Target and Requester.

The auto-initialization/process-execution cycle is repeated until the channel is either disabled or reprogrammed.

Buffer Chaining Process

The Buffer Chaining Process is useful for transferring large quantities of data into non-contiguous buffer areas. In this process, a single channel is used to process data from several buffers, while having to program the channel only once. Each new buffer is programmed in a pipelined operation that provides the new buffer information while the old buffer is being processed. The chain is created by loading new buffer information while the 82370 DMA Controller is processing the Current Buffer. When the Current Buffer expires, the 82370 DMA Controller automatically restarts the channel using the new buffer information.

Loading the new buffer information is done by an interrupt routine which is requested by the 82370. Interrupt Request 1 (IRQ1) is tied internally to the 82370 DMA Controller for this purpose. IRQ1 is generated by the 82370 when the new buffer information is loaded into the channel's Current Registers, leaving the Base Registers 'empty'. The interrupt service routine loads new buffer information into the Base Registers. The host processor is required to load the information for another buffer before the current Byte Count expires. The process repeats until the host programs the channel back to single buffer operation, or until the channel runs out of buffers.

The channel runs out of buffers when the Current Buffer expires and the Base Registers have not yet been loaded with new buffer information. When this occurs, the channel must be reprogrammed.

If an external EOP# is encountered while executing a Buffer Chaining Process, the current buffer is considered expired and the new buffer information is loaded into the Current Registers. If the Base Registers are 'empty', the chain is terminated.

The channel uses the Base Target Address Register as an indicator of whether or not the Base Registers are full. When the most significant byte of the Base Target Register is loaded, the channel considers all of the Base Registers loaded, and removes the interrupt request. This requires that the other Base Registers (Base Requester Address, Base Byte Count) must be loaded before the Base Target Address Register. The reason for implementing the reloading process this way is that, for most applications, the Byte Count and the Requester will not change from one buffer to the next, and therefore do not need to be reprogrammed. The details of programming the channel for the Buffer Chaining Process can be found in the section on DMA programming.

3.3.3 DATA TRANSFER MODES

Three Data Transfer modes are available in the 82370 DMA Controller. They are the Single Transfer, Block Transfer, and Demand Transfer Modes. These transfer modes can be used in conjunction with any one of three Buffer Transfer modes: Single Buffer, Auto-Initialized Buffer and Buffer Chaining. Any Data Transfer Mode can be used under any of the Buffer Transfer Modes. These modes are independently available for all DMA channels.

Different devices being serviced by the DMA Controller require different handshaking sequences for data transfers to take place. Three handshaking modes are available on the 82370, giving the designer the opportunity to use the DMA Controller as efficiently as possible. The speed at which data can

be presented or read by a device can affect the way a DMA Controller uses the host's bus, thereby affecting not only data throughput during the DMA process, but also affecting the host's performance by limiting its access to the bus.

Single Transfer Mode

In the Single Transfer Mode, one data transfer to or from the Requester is performed by the DMA Controller at a time. The DREQn input is arbitrated and the HOLD/HLDA sequence is executed for each transfer. Transfers continue in this manner until the Byte Count expires, or until EOP# is sampled active. If the DREQn input is held active continuously, the entire DREQ-HOLD-HLDA-DACK sequence is repeated over and over until the programmed number of bytes has been transferred. Bus control is released to the host between each transfer. Figure 3-4 shows the logical flow of events which make up a buffer transfer using the Single Transfer Mode. Refer to section 3.4 for an explanation of the bus control arbitration procedure.

The Single Transfer Mode is used for devices which require complete handshake cycles with each data access. Data is transferred to or from the Requester only when the Requester is ready to perform the transfer. Each transfer requires the entire DREQ-

HOLD-HLDA-DACK handshake cycle. Figure 3-5 shows the timing of the Single Transfer Mode cycle.

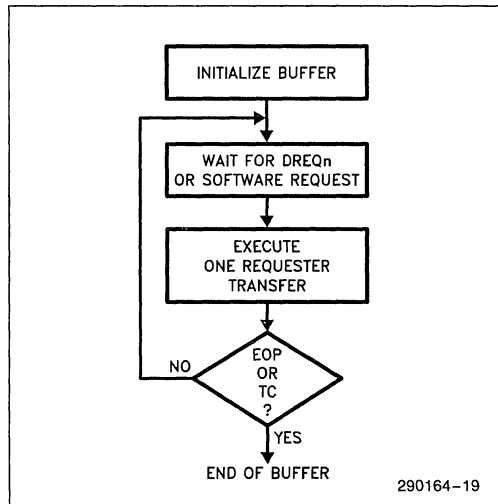
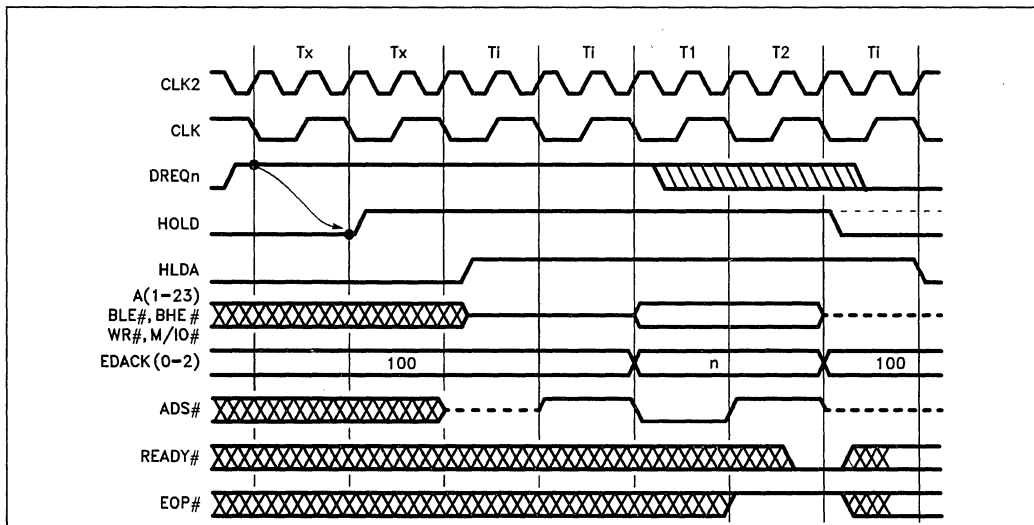


Figure 3-4. Buffer Transfer in Single Transfer Mode



NOTE:

The Single Transfer Mode is more efficient (15%–20%) in the case where the source is the Target. Because of the internal pipeline of the 82370 DMA Controller, two idle states are added at the end of a transfer in the case where the source is the Requester.

Figure 3-5. DMA Single Transfer Mode

Block Transfer Mode

In the Block Transfer Mode, the DMA process is initiated by a DMA request and continues until the Byte Count expires, or until EOP# is activated by the Requester. The DREQn signal need only be held active until the first Requester access. Only a refresh cycle will interrupt the block transfer process.

Figure 3-6 illustrates the operation of the DMA during the Block Transfer Mode. Figure 3-7 shows the timing of the handshake signals during Block Mode Transfers.

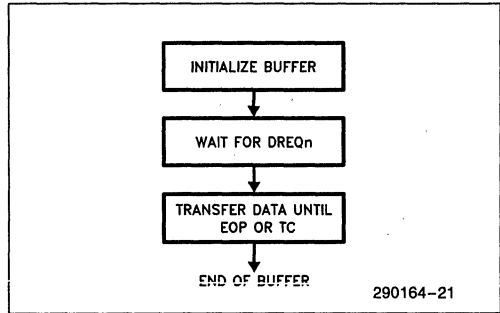


Figure 3-6. Buffer Transfer in Block Transfer Mode

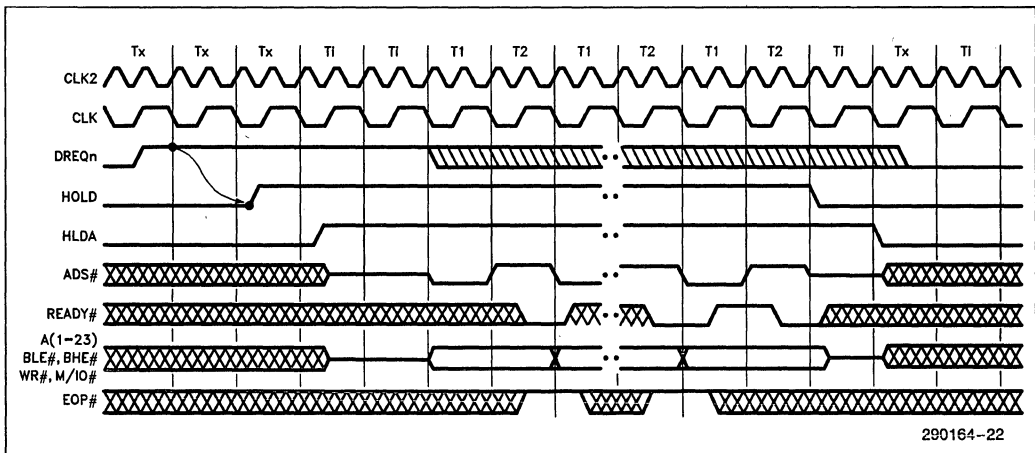


Figure 3-7. Block Mode Transfers

Demand Transfer Mode

The Demand Transfer Mode provides the most flexible handshaking procedures during the DMA process. A Demand Transfer is initiated by a DMA request. The process continues until the Byte Count expires, or an external EOP# is encountered. If the device being serviced (Requester) desires, it can interrupt the DMA process by de-activating the DREQn line. Action is taken on the condition of DREQn during Requester accesses only. The access during which DREQn is sampled inactive is the last Requester access which will be performed during the current transfer. Figure 3-8 shows the flow of events during the transfer of a buffer in the Demand Mode.

When the DREQn line goes inactive, the DMA Controller will complete the current transfer, including any necessary accesses to the Target, and relinquish control of the bus to the host. The current process information is saved (byte count, Requester and Target addresses, and Temporary Register).

The Requester can restart the transfer process by reasserting DREQn. The 82370 will arbitrate the request with other pending requests and begin the process where it left off. Figure 3-9 shows the timing of handshake signals during Demand Transfer Mode operation.

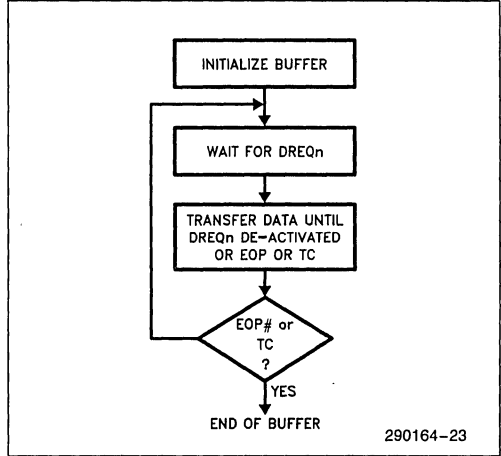


Figure 3-8. Buffer Transfer in Demand Transfer Mode

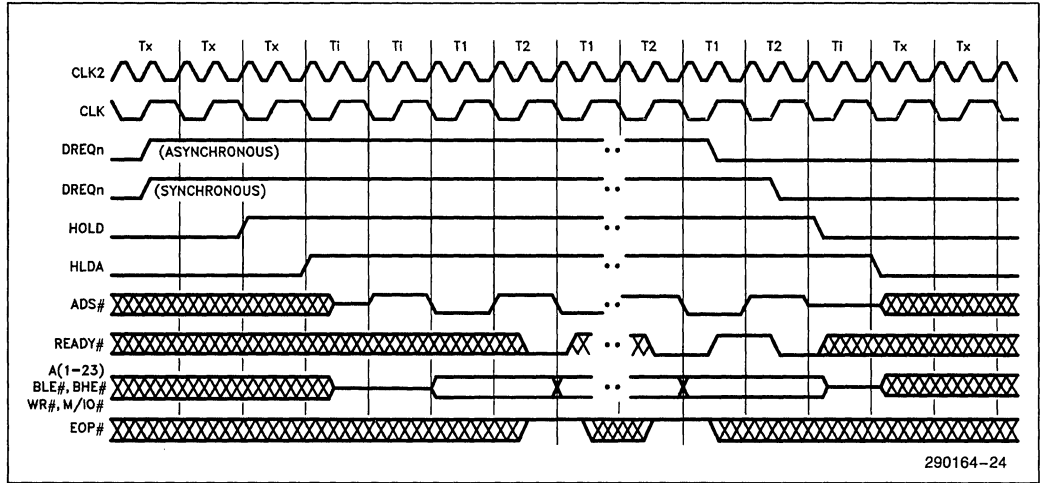


Figure 3-9. Demand Mode Transfers

Using the Demand Transfer Mode allows peripherals to access memory in small, irregular bursts without wasting bus control time. The 82370 is designed to give the best possible bus control latency in the Demand Transfer Mode. Bus control latency is defined here as the time from the last active bus cycle of the previous bus master to the first active bus cycle of the new bus master. The 82370 DMA Controller will perform its first bus access cycle two bus states after HLDA goes active. In the typical configuration, bus control is returned to the host one bus state after the DREQn goes inactive.

There are two cases where there may be more than one bus state of bus control latency at the end of a transfer. The first is at the end of an Auto-Initialize process, and the second is at the end of a process where the source is the Requester and Two-Cycle transfers are used.

When a Buffer Auto-Initialize Porcess is complete, the 82370 requires seven bus states to reload the Current Registers from the Base Registers of the Auto-Initialized channel. The reloading is done while the 82370 is still the bus master so that it is prepared to service the channel immediately after relinquishing the bus, if necessary.

In the case where the Requester is the source, and Two-Cycle transfers are being used, there are two extra idle states at the end of the transfer process. This occurs due to the housekeeping in the DMA's internal pipeline. These two idle states are present only after the very last Requester access, before the DMA Controller de-activates the HOLD signal.

3.3.4 CHANNEL PRIORITY ARBITRATION

DMA channel priority can be programmed into one of two arbitration methods: Fixed or Rotating. The four lower DMA channels and the four upper DMA channels operate as if they were two separate DMA controllers operating in cascade. The lower group of four channels (0-3) is always prioritized between channels 7 and 4 of the upper group of channels (4-7). Figure 3-10 shows a pictorial representation of the priority grouping.

The priority can thus be set up as rotating for one group of channels and fixed for the other, or any other combination. While in Fixed Priority, the programmer can also specify which channel has the lowest priority.

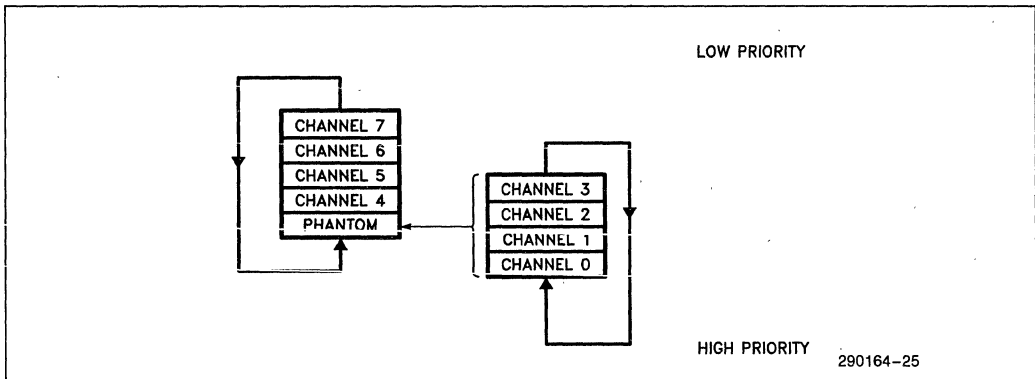


Figure 3-10. DMA Priority Grouping

The 82370 DMA Controller defaults to Fixed Priority. Channel 0 has the highest priority, then 1, 2, 3, 4, 5, 6, 7. Channel 7 has the lowest priority. Any time the DMA Controller arbitrates DMA requests, the requesting channel with the highest priority will be serviced next.

Fixed Priority can be entered into at any time by a software command. The priority levels in effect after the mode switch are determined by the current setting of the Programmable Priority.

Programmable Priority is available for fixing the priority of the DMA channels within a group to levels other than the default. Through a software command, the channel to have the lowest priority in a group can be specified. Each of the two groups of four channels can have the priority fixed in this way. The other channels in the group will follow the natural Fixed Priority sequence. This mode affects only the priority levels while operating with Fixed Priority.

For example, if channel 2 is programmed to have the lowest priority in its group, channel 3 has the highest priority. In descending order, the other channels would have the following priority: (3,0,1,2),4,5,6,7 (channel 2 lowest, channel 3 highest). If the upper

group were programmed to have channel 5 as the lowest priority channel, the priority would be (again, highest to lowest): 6,7, (3,0,1,2), 4,5. Figure 3-11 shows this example pictorially. The lower group is always prioritized as a fifth channel of the upper group (between channels 4 and 7).

The DMA Controller will only accept Programmable Priority commands while the addressed group is operating in Fixed Priority. Switching from Fixed to Rotating Priority preserves the current priority levels. Switching from Rotating to Fixed Priority returns the priority levels to those which were last programmed by use of Programmable Priority.

Rotating Priority allows the devices using DMA to share the system bus more evenly. An individual channel does not retain highest priority after being serviced, priority is passed to the next highest priority channel in the group. The channel which was most recently serviced inherits the lowest priority. This rotation occurs each time a channel is serviced. Figure 3-12 shows the sequence of events as priority is passed between channels. Note that the lower group rotates within the upper group, and that servicing a channel within the lower group causes rotation within the group as well as rotation of the upper group.

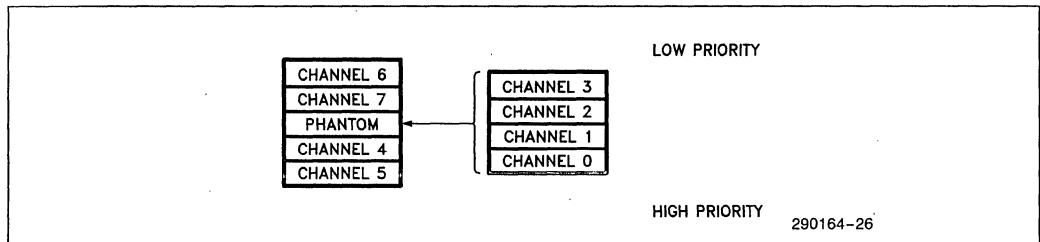


Figure 3-11. Example of Programmed Priority

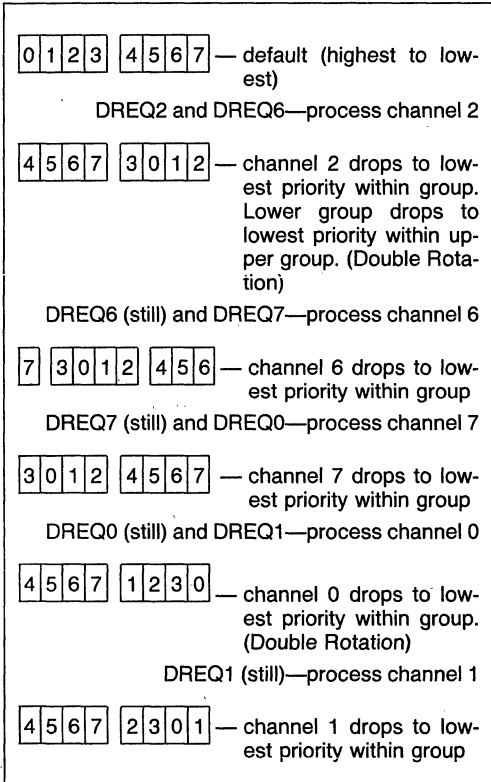


Figure 3-12. Rotating Channel Priority.
 Lower and upper groups are programmed for the Rotating Priority Mode.

3.3.5 COMBINING PRIORITY MODES

Since the DMA Controller operates as two four-channel controllers in cascade, the overall priority scheme of all eight channels can take on a variety of forms. There are four possible combinations of priority modes between the two groups of channels: Fixed Priority only (default), Fixed Priority upper group/Rotating Priority lower group, Rotating Priority upper group/Fixed Priority lower group, and Rotating Priority only. Figure 3-13 illustrates the operation of the two combined priority methods.

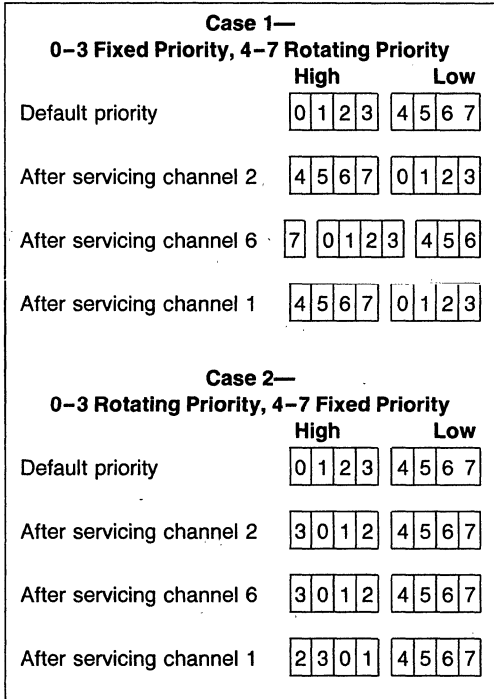


Figure 3-13. Combining Priority Modes

3.3.6 BUS OPERATION

Data may be transferred by the DMA Controller using two different bus cycle operations: Fly-By (one-cycle) and Two-Cycle. These bus handshake methods are selectable independently for each channel through a command register. Device data path widths are independently programmable for both Target and Requester. Also selectable through software is the direction of data transfer. All of these parameters affect the operation of the 82370 on a bus-cycle by bus-cycle basis.

3.3.6.1 Fly-By Transfers

The Fly-By Transfer Mode is the fastest and most efficient way to use the 82370 DMA Controller to transfer data. In this method of transfer, the data is written to the destination device at the same time it is read from the source. Only one bus cycle is used to accomplish the transfer.

In the Fly-By Mode, the DMA acknowledge signal is used to select the Requester. The DMA Controller simultaneously places the address of the Target on the address bus. The state of M/IO# and W/R# during the Fly-By transfer cycle indicate the type of Target and whether the Target is being written to or read from. The Target's Bus Size is used as an incrementer for the Byte Count. The Requester address registers are ignored during Fly-By transfers.

Note that memory-to-memory transfers cannot be done using the Fly-By Mode. Only one memory of I/O address is generated by the DMA Controller at a time during Fly-By transfers. Only one of the devices being accessed can be selected by an address. Also, the Fly-By method of data transfer limits the hardware to accesses of devices with the same data bus width. The Temporary Registers are not affected in the Fly-By Mode.

Fly-By transfers also require that the data paths of the Target and Requester be directly connected. This requires that successive Fly-By access be to word boundaries, or that the Requester be capable of switching its connections to the data bus.

3.3.6.2. Two-Cycle Transfers

Two-Cycle transfers can also be performed by the 82370 DMA Controller. These transfers require at least two bus cycles to execute. The data being transferred is read into the DMA Controller's Temporary Register during the first bus cycle(s). The second bus cycle is used to write the data from the Temporary Register to the destination.

If the addresses of the data being transferred are not word aligned, the 82370 will recognize the situation and read and write the data in groups of bytes, placing them always at the proper destination. This process of collecting the desired bytes and putting them together is called "byte assembly". The reverse process (reading from aligned locations and writing to non-aligned locations) is called "byte disassembly".

The assembly/disassembly process takes place transparent to the software, but can only be done while using the Two-Cycle transfer method. The 82370 will always perform the assembly/disassembly process as necessary for the current data transfer. Any data path widths for either the Requester or Target can be used in the Two-Cycle Mode. This is very convenient for interfacing existing 8- and 16-bit peripherals to the 80376's 16-bit bus.

The 82370 DMA Controller always reads and write data within the word boundaries; i.e. if a word to be

read is crossing a word boundary, the DMA Controller will perform two read operations, each reading one byte, to read the 16-bit word into the Temporary Register. Also, the 82370 DMA Controller always attempts to fill the Temporary Register from the source before writing any data to the destination. If the process is terminated before the Temporary Register is filled (TC or EOP#), the 82370 will write the partial data to the destination. If a process is temporarily suspended (such as when DREQn is deactivated during a demand transfer), the contents of a partially filled Temporary Register will be stored within the 82370 until the process is restarted.

For example, if the source is specified as an 8-bit device and the destination as a 32-bit device, there will be four reads as necessary from the 8-bit source to fill the Temporary Register. Then the 82370 will write the 32-bit contents to the destination in two cycles of 16-bit each. This cycle will repeat until the process is terminated or suspended.

With Two-Cycle transfers, the devices that the 82370 accesses can reside at any address within I/O or memory space. The device must be able to decode the byte-enables (BLE#, BHE#). Also, if the device cannot accept data in byte quantities, the programmer must take care not to allow the DMA Controller to access the device on any address other than the device boundary.

3.3.6.3 Data Path Width and Data Transfer Rate Considerations

The number of bus cycles used to transfer a single "word" of data is affected by whether the Two-Cycle or the Fly-By (Single-Cycle) transfer method is used.

The number of bus cycles used to transfer data directly affects the data transfer rate. Inefficient use of bus cycles will decrease the effective data transfer rate that can be obtained. Generally, the data transfer rate is halved by using Two-Cycle transfers instead of Fly-By transfers.

The choice of data path widths of both Target and Requester affects the data transfer rate also. During each bus cycle, the largest pieces of data possible should be transferred.

The data path width of the devices to be accessed must be programmed into the DMA controller. The 82370 defaults after reset to 8-bit-to-8-bit data transfers, but the Target and Requester can have different data path widths, independent of each other and independent of the other channels. Since this is a software programmable function, more discussion of the uses of this feature are found in the section on programming.

3.3.6.4 Read, Write and Verify Cycles

Three different bus cycles types may be used in a data transfer. They are the Read, Write and Verify cycles. These cycle types dictate the way in which the 82370 operates on the data to be transferred.

A Read Cycle transfers data from the Target to the Requester. A Write Cycle transfers data from the Requester to the target. In a Fly-By transfer, the address and bus status signals indicate the access (read or write) to the Target; the access to the Requester is assumed to be the opposite.

The Verify Cycle is used to perform a data read only. No write access is indicated or assumed in a Verify Cycle. The Verify Cycle is useful for validating block fill operations. An external comparator must be provided to do any comparisons on the data read.

3.4 Bus Arbitration and Handshaking

Figure 3-14 shows the flow of events in the DMA request arbitration process. The arbitration sequence starts when the Requester asserts a DREQn (or DMA service is requested by software). Figure 3-15 shows the timing of the sequence of events following a DMA request. This sequence is executed for each channel that is activated. The DREQn signal can be replaced by a software DMA channel request with no change in the sequence.

After the Requester asserts the service request, the 82370 will request control of the bus via the HOLD signal. The 82370 will always assert the HOLD signal one bus state after the service request is asserted. The 80376 responds by asserting the HLDA signal, thus releasing control of the bus to the 82370 DMA Controller.

Priority of pending DMA service requests is arbitrated during the first state after HLDA is asserted by the 80376. The next state will be the beginning of the first transfer access of the highest priority process.

When the 82370 DMA Controller is finished with its current bus activity, it returns control of the bus to the host processor. This is done by driving the HOLD signal inactive. The 82370 does not drive any address or data bus signals after HOLD goes low. It enters the Slave Mode until another DMA process is requested. The processor acknowledges that it has

regained control of the bus by forcing the HLDA signal inactive. Note that the 82370's DMA Controller will not re-request control of the bus until the entire HOLD/HLDA handshake sequence is complete.

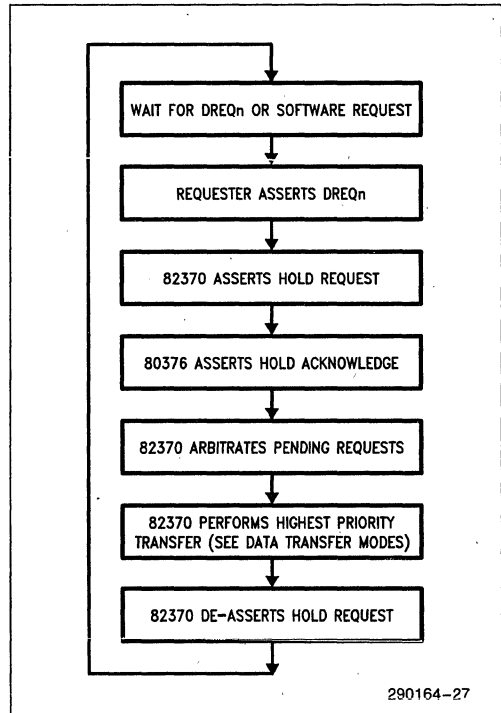


Figure 3-14. Bus Arbitration and DMA Sequence

The 82370 DMA Controller will terminate a current DMA process for one of three reasons: expired byte count, end-of-process command (EOP# activated) from a peripheral, or deactivated DMA request signal. In each case, the controller will de-assert HOLD immediately after completing the data transfer in progress. These three methods of process termination are illustrated in Figures 3-16, 3-19 and 3-18, respectively.

An expired byte count indicates that the current process is complete as programmed and the channel has no further transfers to process. The channel must be restarted according to the currently programmed Buffer Transfer Mode, or reprogrammed completely, including a new Buffer Transfer Mode.

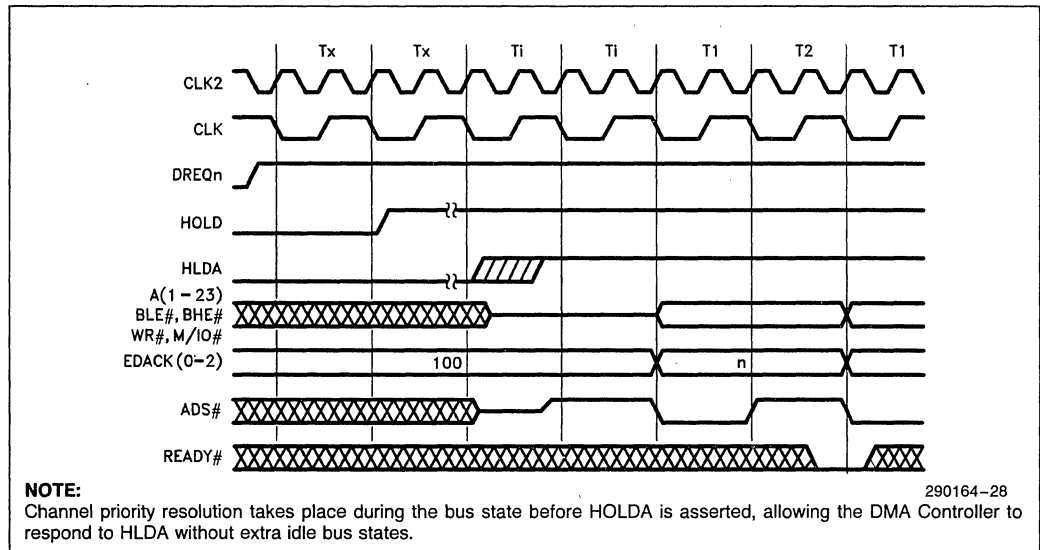


Figure 3-15. Beginning of a DMA process

If the peripheral activates the EOP# signal, it is indicating that it will not accept or deliver any more data for the current buffer. The 82370 DMA Controller considers this as a completion of the channel's current process and interprets the condition the same way as if the byte count expired.

The action taken by the 82370 DMA Controller in response to a de-activated DREQn signal depends on the Data Transfer Mode of the channel. In the Demand Mode, data transfers will take place as long as the DREQn is active and the byte count has not expired. In the Block Mode, the controller will complete the entire block transfer without relinquishing the bus, even if DREQn goes inactive before the

transfer is complete. In the Single Mode, the controller will execute single data transfers, relinquishing the bus between each transfer, as long as DREQn is active.

Normal termination of a DMA process due to expiration of the byte count (Terminal Count—TC) is shown in Figure 3-16. The condition of DREQn is ignored until after the process is terminated. If the channel is programmed to auto-initialize, HOLD will be held active for an additional seven clock cycles while the auto-initialization takes place.

Table 3-3 shows the DMA channel activity due to EOP# or Byte Count expiring (Terminal Count).

Table 3-3. DMA Channel Activity Due to Terminal Count or External EOP#

Buffer Process	Single or Chaining-Base Empty		Auto-Initialize		Chaining-Base Loaded	
	True	X	True	X	True	X
EVENT						
Terminal Count	True	X	True	X	True	X
EOP#	X	0	X	0	X	0
RESULTS						
Current Registers			Load	Load	Load	Load
Channel Mask	Set	Set				
EOP# Output	0	X	0	X	1	X
Terminal Count Status	Set	Set	Set	Set		
Software Request	CLR	CLR	CLR	CLR		

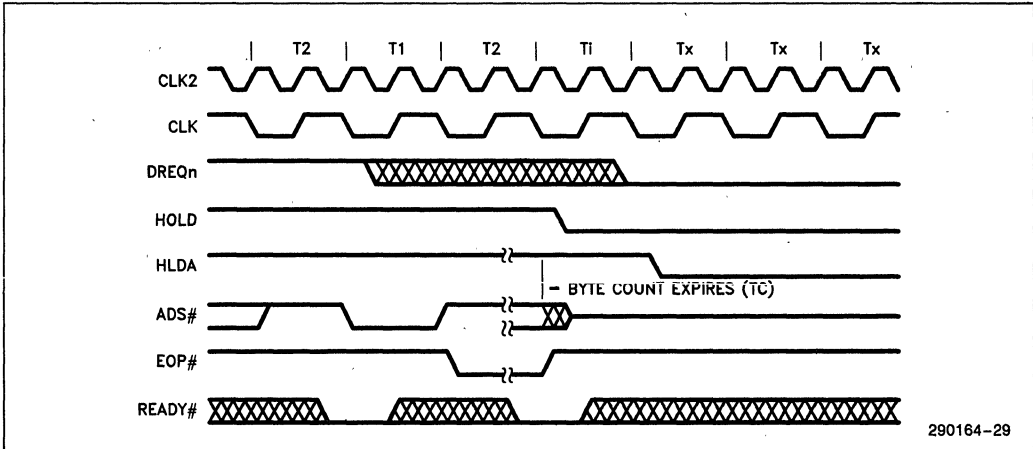


Figure 3-16. Termination of a DMA Process Due to Expiration of Current Byte Count

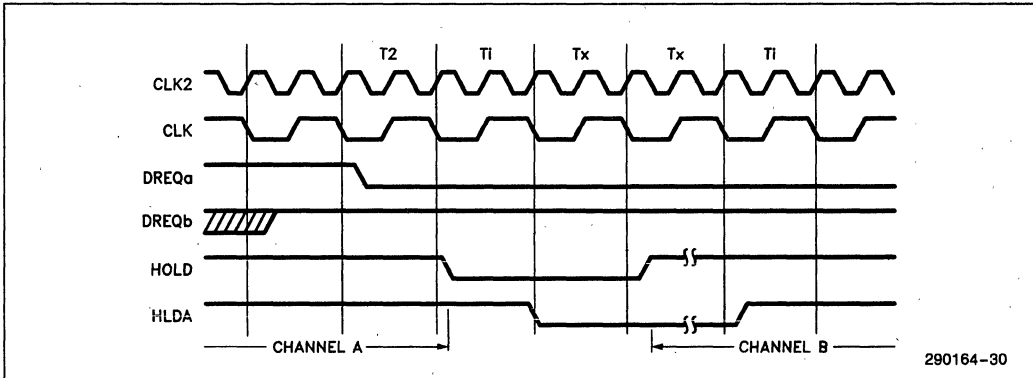


Figure 3-17. Switching between Active DMA Channels

The 82370 always relinquishes control of the bus between channel services. This allows the hardware designer the flexibility to externally arbitrate bus hold requests, if desired. If another DMA request is pending when a higher priority channel service is completed, the 82370 will relinquish the bus until the hold acknowledge is inactive. One bus state after the HLDA signal goes inactive, the 82370 will assert HOLD again. This is illustrated in Figure 3-17.

3.4.1 SYNCHRONOUS AND ASYNCHRONOUS SAMPLING OF DREQn AND EOP#

As an indicator that a DMA service is to be started, DREQn is always sampled asynchronous. It is sam-

pled at the beginning of a bus state and acted upon at the end of the state. Figure 3-15 illustrates the start of a DMA process due to a DREQn input.

The DREQn and EOP# inputs can be programmed to be sampled either synchronously or asynchronously to signal the end of a transfer.

The synchronous mode affords the Requester one bus state of extra time to react to an access. This means the Requester can terminate a process on the current access, without losing any data. The asynchronous mode requires that the input signal be presented prior to the beginning of the last state of the Requester access.

The timing relationships of the DREQn and EOP# signals to the termination of a DMA transfer are shown in Figures 3-18 and 3-19. Figure 3-18 shows the termination of a DMA transfer due to inactive DREQn. Figure 3-19 shows the termination of a DMA process due to an active EOP# input.

In the Synchronous Mode, DREQn and EOP# are sampled at the end of the last state of every Requester data transfer cycle. If EOP# is active or DREQn is inactive at this time, the 82370 recognizes this access to the Requester as the last transfer. At this point, the 82370 completes the transfer in progress, if necessary, and returns bus control to the host.

In the asynchronous mode, the inputs are sampled at the beginning of every state of a Requester access. The 82370 waits until the end of the state to act on the input.

DREQn and EOP# are sampled at the latest possible time when the 82370 can determine if another transfer is required. In the Synchronous Mode, DREQn and EOP# are sampled on the trailing edge of the last bus state before another data access cycle begins. The Asynchronous Mode requires that the signals be valid one clock cycle earlier.

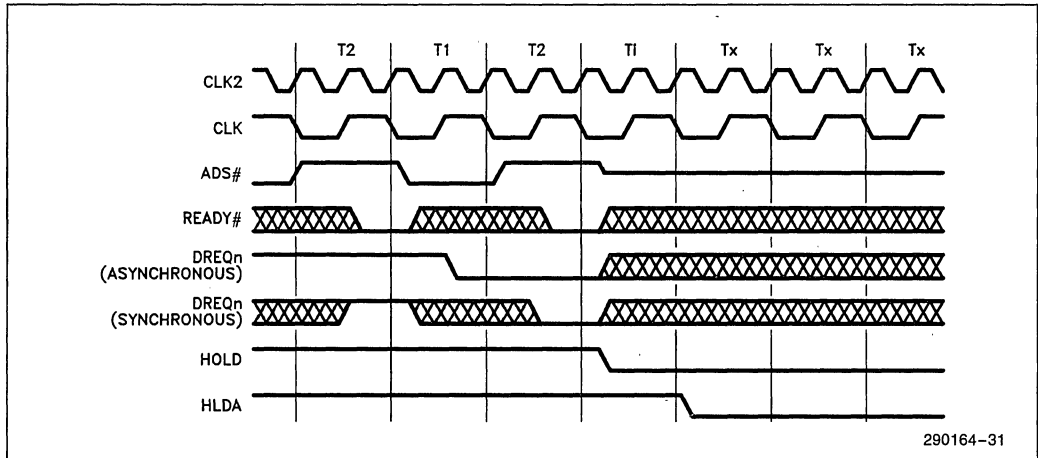


Figure 3-18. Termination of a DMA Process due to De-Asserting DREQn

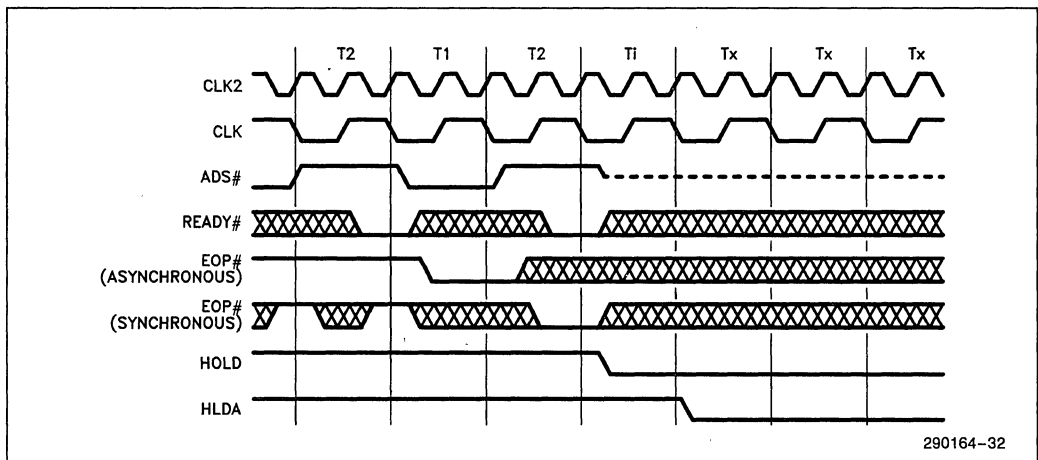


Figure 3-19. Termination of a DMA Process due to an External EOP#

While in the Pipeline Mode, if the NA# signal is sampled active during a transfer, the end of the state where NA# was sampled active is when the 82370 decides whether to commit to another transfer. The device must de-assert DREQn or assert EOP# before NA# is asserted, otherwise the 82370 will commit to another, possibly undesired, transfer.

Synchronous DREQn and EOP# sampling allows the peripheral to prevent the next transfer from occurring by de-activating DREQn or asserting EOP# during the current Requester access, before the 82370 DMA Controller commits itself to another transfer. The DMA Controller will not perform the next transfer if it has not already begun the bus cycle. Asynchronous sampling allows less stringent timing requirements than the Synchronous Mode, but requires that the DREQn signal be valid at the beginning of the next to last bus state of the current Requester access.

Using the Asynchronous Mode with zero wait states can be very difficult. Since the addresses and control signals are driven by the 82370 near half-way through the first bus state of a transfer, and the Asynchronous Mode requires that DREQn be inactive before the end of the state, the peripheral being accessed is required to present DREQn only a few nanoseconds after the control information is available. This means that the peripheral's control logic must be extremely fast (practically non-causal). An alternative is the Synchronous Mode.

3.4.2 ARBITRATION OF CASCADED MASTER REQUESTS

The Cascade Mode allows another DMA-type device to share the bus by arbitrating its bus accesses with the 82370's. Seven of the eight DMA channels (0-3 and 5-7) can be connected to a cascaded device. The cascaded device requests bus control through the DREQn line of the channel which is programmed to operate in Cascade Mode. Bus hold acknowledge is signalled to the cascaded device through the EDACK lines. When the EDACK lines are active with the code for the requested cascade channel, the bus is available to the cascaded master device.

A cascade cycle begins the same way a regular DMA cycle begins. The requesting bus master asserts the DREQn line on the 82370. This bus control request is arbitrated as any other DMA request would be. If any channel receives a DMA request, the 82370 requests control of the bus. When the host acknowledges that it has released bus control, the 82370 acknowledges to the requesting master that it may access the bus. The 82370 enters an idle state until the new master relinquishes control.

A cascade cycle will be terminated by one of two events: DREQn going inactive, or HLDA going inactive. The normal way to terminate the cascade cycle

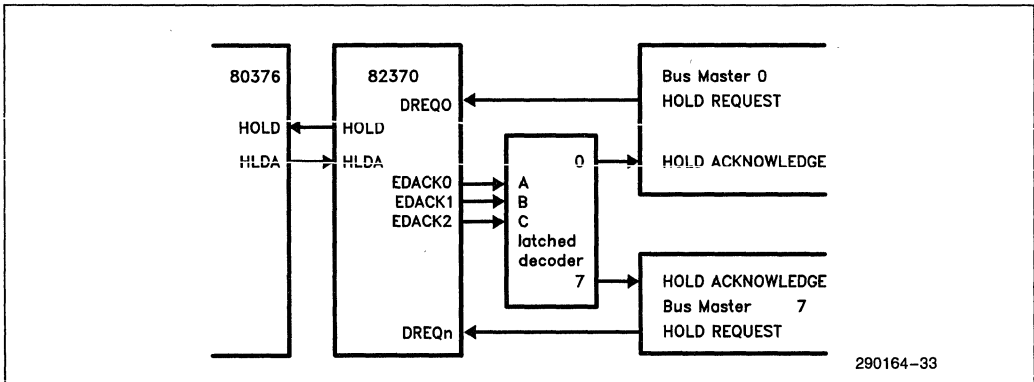


Figure 3-20. Cascaded Bus Master

is for the cascaded master to drop the DREQn signal. Figure 3-21 shows the two cascade cycle termination sequences.

The Refresh Controller may interrupt the cascaded master to perform a refresh cycle. If this occurs, the 82370 DMA Controller will de-assert the EDACK signal (hold acknowledge to cascaded master) and wait for the cascaded master to remove its hold request. When the 82370 regains bus control, it will perform the refresh cycle in its normal fashion. After the refresh cycle has been completed, and if the cascaded device has re-asserted its request, the 82370 will return control to the cascaded master which was interrupted.

The 82370 assumes that it is the only device monitoring the HLDA signal. If the system designer wishes to place other devices on the bus as bus masters, the HLDA from the processor must be intercepted before presenting it to the 82370. Using the Cascade capability of the 82370 DMA Controller offers a much better solution.

3.4.3 ARBITRATION OF REFRESH REQUESTS

The arbitration of refresh requests by the DRAM Refresh Controller is slightly different from normal DMA

channel request arbitration. The 82370 DRAM Refresh Controller always has the highest priority of any DMA process. It also can interrupt a process in progress. Two types of processes in progress may be encountered: normal DMA, and bus master cascade.

In the event of a refresh request during a normal DMA process, the DMA Controller will complete the data transfer in progress and then execute the refresh cycle before continuing with the current DMA process. The priority of the interrupted process is not lost. If the data transfer cycle interrupted by the Refresh Controller is the last of a DMA process, the refresh cycle will always be executed before control of the bus is transferred back to the host.

When the Refresh Controller request occurs during a cascade cycle, the Refresh Controller must be assured that the cascaded master device has relinquished control of the bus before it can execute the refresh cycle. To do this, the DMA Controller drops the EDACK signal to the cascaded master and waits for the corresponding DREQn input to go inactive. By dropping the DREQn signal, the cascaded master relinquishes the bus. The Refresh Controller then performs the refresh cycle. Control of the bus is returned to the cascaded master if DREQn returns to an active state before the end of the refresh cycle, otherwise control is passed to the processor and the cascaded master loses its priority.

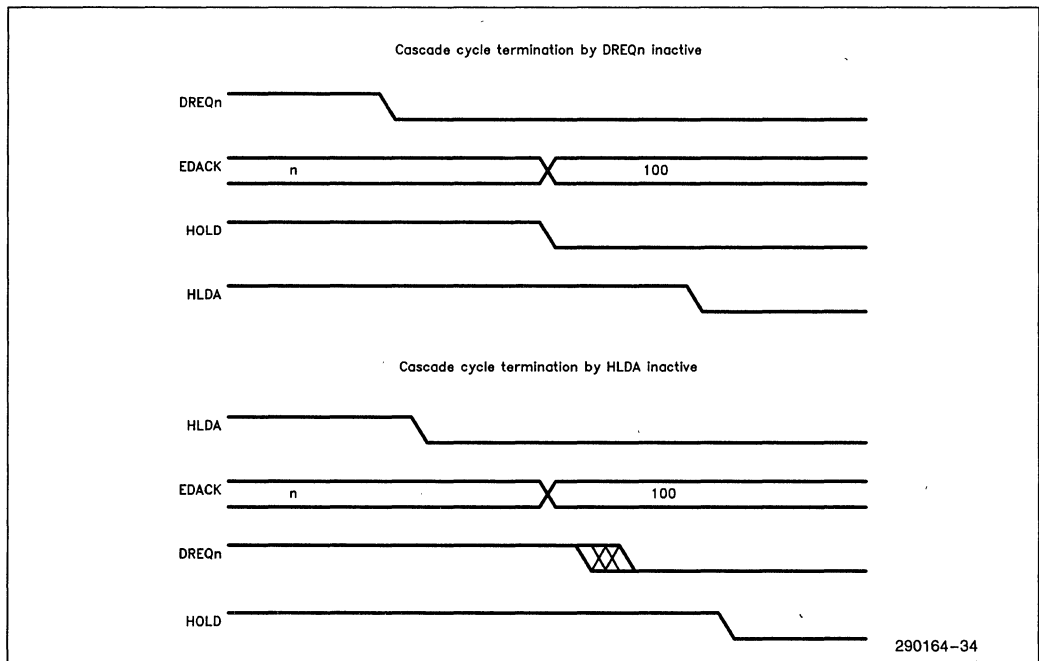


Figure 3-21. Cascade Cycle Termination

3.5 DMA Controller Register Overview

The 82370 DMA Controller contains 44 registers which are accessible to the host processor. Twenty-four of these registers contain the device addresses and data counts for the individual DMA channels (three per channel). The remaining registers are control and status registers for initiating and monitoring the operation of the 82370 DMA Controller. Table 3-4 lists the DMA Controller's registers and their accessibility.

Table 3-4. DMA Controller Registers

Register Name	Access
Control/Status Registers—one each per group	
Command Register I	write only
Command Register II	write only
Mode Register I	write only
Mode Register II	write only
Software Request Register	read/write
Mask Set-Reset Register	write only
Mask Read-Write Register	read/write
Status Register	read only
Bus Size Register	write only
Chaining Register	read/write
Channel Registers—one each per channel	
Base Target Address	write only
Current Target Address	read only
Base Requester Address	write only
Current Requester Address	read only
Base Byte Count	write only
Current Byte Count	read only

3.5.1 CONTROL/STATUS REGISTERS

The following registers are available to the host processor for programming the 82370 DMA Controller into its various modes and for checking the operating status of the DMA processes. Each set of four DMA channels has one of each of these registers associated with it.

Command Register I

Enables or disables the DMA channel as a group. Sets the Priority Mode (Fixed or Rotating) of the group. This write-only register is cleared by a hardware reset, defaulting to all channels enabled and Fixed Priority Mode.

Command Register II

Sets the sampling mode of the DREQn and EOP# inputs. Also sets the lowest priority channel for the group in the Fixed Priority Mode. The functions programmed through Command Register II default after

a hardware reset to: asynchronous DREQn and EOP#, and channels 3 and 7 lowest priority.

Mode Registers I

Mode Register I is identical in function to the Mode register of the 8237A. It programs the following functions for an individually selected channel:

- Type of Transfer—read, write, verify
- Auto-Initialize—enable or disable
- Target Address Count—increment or decrement
- Data Transfer Mode—demand, single, block, cascade

Mode Register I functions default to the following after reset: verify transfer, Auto-Initialize disabled, Increment Target address, Demand Mode.

Mode Register II

Programs the following functions for an individually selected channel:

- Target Address Hold—enable or disable
- Requester Address Count—increment or decrement
- Requester Address Hold—enable or disable
- Target Device Type—I/O or Memory
- Requester Device Type—I/O or Memory
- Transfer Cycles—Two-Cycle or Fly-By

Mode Register II functions are defined as follows after a hardware reset: Disable Target Address Hold, Increment Requester Address, Target (and Requester) in memory, Fly-By Transfer Cycles. Note: Requester Device Type ignored in Fly-By Transfers.

Software Request Register

The DMA Controller can respond to service requests which are initiated by software. Each channel has an internal request status bit associated with it. The host processor can write to this register to set or reset the request bit of a selected channel.

The status of a group's software DMA service requests can be read from this register as well. Each status bit is cleared upon Terminal Count or external EOP#.

The software DMA requests are non-maskable and subject to priority arbitration with all other software and hardware requests. The entire register is cleared by a hardware reset.

Mask Registers

Each channel has associated with it a mask bit which can be set/reset to disable/enable that channel. Two methods are available for setting and clearing the mask bits. The Mask Set/Reset Register is a

write-only register which allows the host to select an individual channel and either set or reset the mask bit for that channel only. The Mask Read/Write Register is available for reading the mask bit status and for writing mask bits in groups of four.

The mask bits of a group may be cleared in one step by executing the Clear Mask Command. See the DMA Programming section for details. A hardware reset sets all of the channel mask bits, disabling all channels.

Status Register

The Status register is a read-only register which contains the Terminal Count (TC) and Service Request status for a group. Four bits indicate the TC status and four bits indicate the hardware request status for the four channels in the group. The TC bits are set when the Byte Count expires, or when and external EOP# is asserted. These bits are cleared by reading from the Status Register. The Service Request bit for a channel indicates when there is a hardware DMA request (DREQn) asserted for that channel. When the request has been removed, the bit is cleared.

Bus Size Register

This write-only register is used to define the bus size of the Target and Requester of a selected channel. The bus sizes programmed will be used to dictate the sizes of the data paths accessed when the DMA channel is active. The values programmed into this register affect the operation of the Temporary Register. When 32-bit bus width is programmed, the 82370 DMA Controller will access the device twice through its 16-bit external Data Bus to perform a 32-bit data transfer. Any byte-assembly required to make the transfers using the specified data path widths will be done in the Temporary Register. The Bus Size register of the Target is used as an increment/decrement value for the Byte Counter and Target Address when in the Fly-By Mode. Upon reset, all channels default to 8-bit Targets and 8-bit Requesters.

Chaining Register

As a command or write register, the Chaining register is used to enable or disable the Chaining Mode for a selected channel. Chaining can either be disabled or enabled for an individual channel, independently of the Chaining Mode status of other channels. After a hardware reset, all channels default to Chaining disabled.

When read by the host, the Chaining Register provides the status of the Chaining Interrupt of each of the channels. These interrupt status bits are cleared when the new buffer information has been loaded.

3.5.2 CHANNEL REGISTERS

Each channel has three individually programmable registers necessary for the DMA process; they are the Base Byte Count, Base Target Address, and Base Requester Address registers. The 24-bit Base Byte Count register contains the number of bytes to be transferred by the channel. The 24-bit Base Target Address Register contains the beginning address (memory or I/O) of the Target device. The 24-bit Base Requester Address register contains the base address (memory or I/O) of the device which is to request DMA service.

Three more registers for each DMA channel exist within the DMA Controller which are directly related to the registers mentioned above. These registers contain the current status of the DMA process. They are the Current Byte Count register, the Current Target Address, and the Current Requester Address. It is these registers which are manipulated (incremented, decremented, or held constant) by the 82370 DMA Controller during the DMA process. The Current registers are loaded from the Base registers at the beginning of a DMA process.

The Base registers are loaded when the host processor writes to the respective channel register addresses. Depending on the mode in which the channel is operating, the Current registers are typically loaded in the same operation. Reading from the channel register addresses yields the contents of the corresponding Current register.

To maintain compatibility with software which accesses an 8237A, a Byte Pointer Flip-Flop is used to control access to the upper and lower bytes of some words of the Channel Registers. These words are accessed as byte pairs at single port addresses. The Byte Pointer Flip-Flop acts as a one-bit pointer which is toggled each time a qualifying Channel Register byte is accessed.

It always points to the next logical byte to be accessed of a pair of bytes.

The Channel registers are arranged as pairs of words, each pair with its own port address. Addressing the port with the Byte Pointer Flip-Flop reset accesses the least significant byte of the pair. The most significant byte is accessed when the Byte Pointer is set.

For compatibility with existing 8237A designs, there is one exception to the above statements about the Byte Pointer Flip-Flop. The third byte (bits 16-23) of the Target Address is accessed through its own port address. The Byte Pointer Flip-Flop is not affected by any accesses to this byte.

The upper eight bits of the Byte Count Register are cleared when the least significant byte of the register is loaded. This provides compatibility with software which accesses an 8237A. The 8237A has 16-bit Byte Count Registers.

NOTE:

The 82370 is a subset of the Intel 82380 32-bit DMA Controller with Integrated System Peripherals.

Although the 82370 has 24 address bits externally, the programming model is actually a full 32 bits wide. For this reason, there are some "hidden" DMA registers in the 82370 register set. These hidden registers correspond to what would be A24-A31 in a 32-bit system.

Think of the 82370 addresses as though they were 32 bits wide, with only the lower 24 bits available externally.

This should be of concern in two areas:

1. Understanding the Byte Pointer Flip Flop
2. Removing the IRQ1 Chaining Interrupt

The byte pointer flip flop will behave as though the hidden upper address bits were accessible.

The IRQ1 Chaining Interrupt will be removed only when the hidden upper address bits are programmed. You will note that since the hidden upper address bits are not available externally, the value you program into the registers is not important. The act of programming the hidden register is critical in removing the IRQ1 Chaining interrupt for a DMA channel.

The port assignments for these hidden upper address bits come directly from the port assignments of the Intel 82380. For your convenience, those port definitions have been included in this data sheet in section 3.7.

3.5.3 TEMPORARY REGISTERS

Each channel has a 32-bit Temporary Register used for temporary data storage during two-cycle DMA transfers. It is this register in which any necessary byte assembly and disassembly of non-aligned data is performed. Figure 3-22 shows how a block of data will be moved between memory locations with different boundaries. Note that the order of the data does not change.

If the destination is the Requester and an early process termination has been indicated by the EOP# signal or DREQn inactive in the Demand Mode, the Temporary Register is not affected. If data remains in the Temporary Register due to differences in data path widths of the Target and Requester, it will not

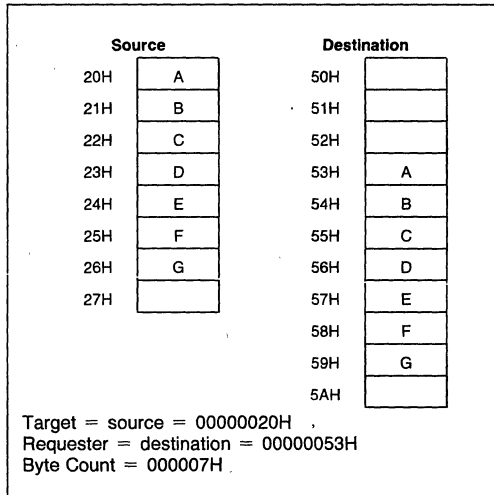


Figure 3-22. Transfer of data between memory locations with different boundaries. This will be the result, independent of data path width.

be transferred or otherwise lost, but will be stored for later transfer.

If the destination is the Target and the EOP# signal is sensed active during the Requester access of a transfer, the DMA Controller will complete the transfer by sending to the Target whatever information is in the Temporary Register at the time of process termination. This implies that the Target could be accessed with partial data in two accesses. For this reason it is advisable to have an I/O device designated as a Requester, unless it is capable of handling partial data transfers.

3.6 DMA Controller Programming

Programming a DMA Channel to perform a needed DMA function is in general a four step process. First the global attributes of the DMA Controller are programmed via the two Command Registers. These global attributes include: priority levels, channel group enables, priority mode, and DREQn/EOP# input sampling.

The second step involves setting the operating modes of the particular channel. The Mode Registers are used to define the type of transfer and the handshaking modes. The Bus Size Register and Chaining Register may also need to be programmed in this step.

The third step in setting up the channel is to load the Base Registers in accordance with the needs of the operating modes chosen in step two. The Current Registers are automatically loaded from the Base Registers, if required by the Buffer Transfer Mode in

effect. The information loaded and the order in which it is loaded depends on the operating mode. A channel used for cascading, for example, needs no buffer information and this step can be skipped entirely.

The last step is to enable the newly programmed channel using one of the Mask Registers. The channel is then available to perform the desired data transfer. The status of the channel can be observed at any time through the Status Register, Mask Register, Chaining Register, and Software Request register.

Once the channel is programmed and enabled, the DMA process may be initiated in one of two ways, either by a hardware DMA request (DREQn) or a software request (Software Request Register).

Once programmed to a particular Process/Mode configuration, the channel will operate in that configuration until programmed otherwise. For this reason, restarting a channel after the current buffer expires does not require complete reprogramming of the channel. Only those parameters which have changed need to be reprogrammed. The Byte Count Register is always changed and must be reprogrammed. A Target or Requester Address Register which is incremented or decremented should be reprogrammed also.

3.6.1 BUFFER PROCESSES

The Buffer Process is determined by the Auto-Initialize bit of Mode Register I and the Chaining Register. If Auto-Initialize is enabled, Chaining should not be used.

3.6.1.1 Single Buffer Process

The Single Buffer Process is programmed by disabling Chaining via the Chaining Register and programming Mode Register I for non-Auto-Initialize.

3.6.1.2 Buffer Auto-Initialize Process

Setting the Auto-Initialize bit in Mode Register I is all that is necessary to place the channel in this mode. Buffer Auto-Initialize must not be enabled simultaneous to enabling the Buffer Chaining Mode as this will have unpredictable results.

Once the Base Registers are loaded, the channel is ready to be enabled. The channel will reload its Current Registers from the Base Registers each time the Current Buffer expires, either by an expired Byte Count or an external EOP#.

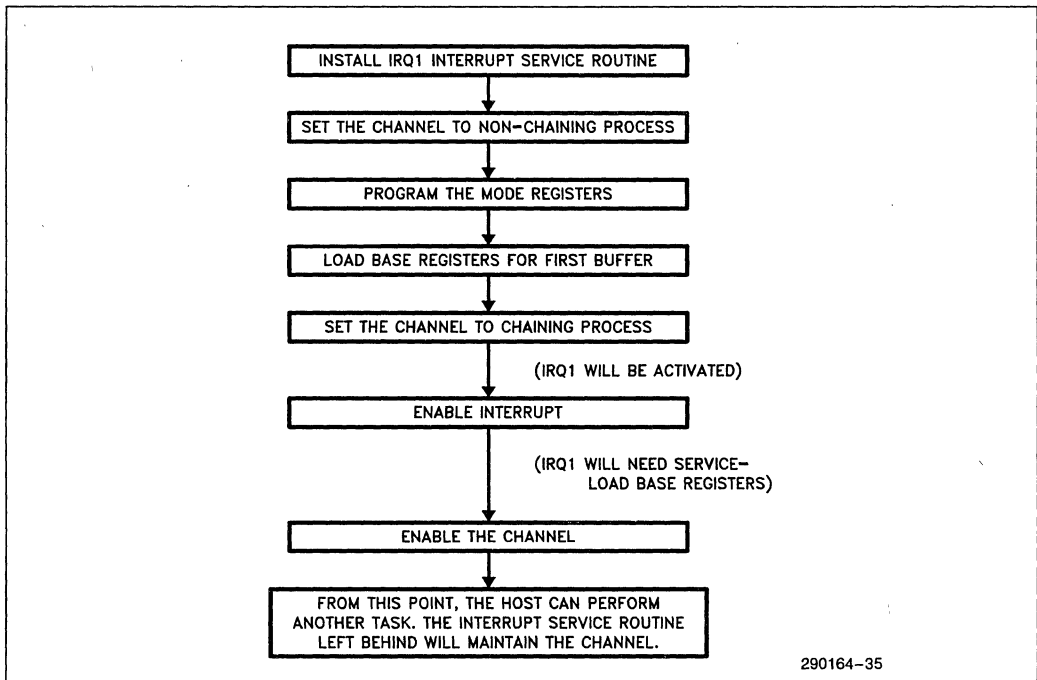


Figure 3-23. Flow of Events in the Buffer Chaining Process

3.6.1.3 Buffer Chaining Process

The Buffer Chaining Process is entered into from the Single Buffer Process. The Mode Registers should be programmed first, with all of the Transfer Modes defined as if the channel were to operate in the Single Buffer Process. The channel's Base Registers are then loaded. When the channel has been set up in this way, and the chaining interrupt service routine is in place, the Chaining Process can be entered by programming the Chaining Register. Figure 3-23 illustrates the Buffer Chaining Process.

An interrupt (IRQ1) will be generated immediately after the Chaining Process is entered, as the channel then perceives the Base Registers as empty and in need of reloading. It is important to have the interrupt service routine in place at the time the Chaining Process is entered into. The interrupt request is removed when the most significant byte of the Base Target Address is loaded.

The interrupt will occur again when the first buffer expires and the Current Registers are loaded from the Base Registers. The cycle continues until the Chaining Process is disabled, or the host fails to respond to IRQ1 before the Current Buffer expires.

Exiting the Chaining Process can be done by resetting the Chaining Mode Register. If an interrupt is pending for the channel when the Chaining Register is reset, the interrupt request will be removed. The Chaining Process can be temporarily disabled by setting the channel's Mask bit in the Mask Register.

The interrupt service routine for IRQ1 has the responsibility of reloading the Base Registers as necessary. It should check the status of the channel to determine the cause of channel expiration, etc. It should also have access to operating system information regarding the channel, if any exists. The IRQ1 service routine should be capable of determining whether the chain should be continued or terminated and act on that information.

3.6.2 DATA TRANSFER MODES

The Data Transfer Modes are selected via Mode Register I. The Demand, Single, and Block Modes are selected by bits D6 and D7. The individual transfer type (Fly-By vs Two-Cycle, Read-Write-Verify, and I/O vs Memory) is programmed through both of the Mode registers.

3.6.3 CASCADED BUS MASTERS

The Cascade Mode is set by writing ones to D7 and D6 of Mode Register I. When a channel is pro-

grammed to operate in the Cascade Mode, all of the other modes associated with Mode Registers I and II are ignored. The priority and DREQn/EOP# definitions of the Command Registers will have the same effect on the channel's operation as any other mode.

3.6.4 SOFTWARE COMMANDS

There are five port addresses which, when written to, command certain operations to be performed by the 82370 DMA Controller. The data written to these locations is not of consequence, writing to the location is all that is necessary to command the 82370 to perform the indicated function. Following are descriptions of the command functions.

Clear Byte Pointer Flip-Flop—Location 000CH

Resets the Byte Pointer Flip-Flop. This command should be performed at the beginning of any access to the channel registers in order to be assured of beginning at a predictable place in the register programming sequence.

Master Clear—Location 000DH

All DMA functions are set to their default states. This command is the equivalent of a hardware reset to the DMA Controller. Functions other than those in the DMA Controller section of the 82370 are not affected by this command.

Clear Mask Register—Channels 0–3
 — Location 000EH
 Channels 4–7
 — Location 00CEH

This command simultaneously clears the Mask Bits of all channels in the addressed group, enabling all of the channels in the group.

Clear TC Interrupt Request—Location 001EH

This command resets the Terminal Count Interrupt Request Flip-Flop. It is provided to allow the program which made a software DMA request to acknowledge that it has responded to the expiration of the requested channel(s).

3.7 Register Definitions

The following diagrams outline the bit definitions and functions of the 82370 DMA Controller's Status and Control Registers. The function and programming of the registers is covered in the previous section on DMA Controller Programming. An entry of "X" as a bit value indicates "don't care."

Channel Registers (read Current, write Base)

Channel	Register Name	Address (hex)	Byte Pointer	Bits Accessed
Channel 0	Target Address	00	0	0-7
			1	8-15
		87	x	16-23
	Byte Count	10	0	24-31(*)
		01	0	0-7
			1	8-15
	Requester Address	11	0	16-23
		90	0	0-7
			1	8-15
		91	0	16-23
			1	24-31(*)
Channel 1	Target Address	02	0	0-7
			1	8-15
		83	x	16-23
	Byte Count	12	0	24-31(*)
		03	0	0-7
			1	8-15
	Requester Address	13	0	16-23
		92	0	0-7
			1	8-15
		93	0	16-23
			1	24-31(*)
Channel 2	Target Address	04	0	0-7
			1	8-15
		81	x	16-23
	Byte Count	14	0	24-31(*)
		05	0	0-7
			1	8-15
	Requester Address	15	0	16-23
		94	0	0-7
			1	8-15
		95	0	16-23
			1	24-31(*)
Channel 3	Target Address	06	0	0-7
			1	8-15
		82	x	16-23
	Byte Count	16	0	24-31(*)
		07	0	0-7
			1	8-15
	Requester Address	17	0	16-23
		96	0	0-7
			1	8-15
		97	0	16-23
			1	24-31(*)

Channel Registers (read Current, write Base) (Continued)

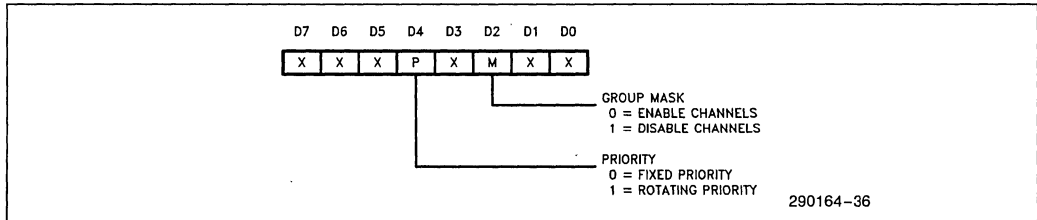
Channel	Register Name	Address (hex)	Byte Pointer	Bits Accessed
Channel 4	Target Address	C0	0	0-7
			1	8-15
		8F	x	16-23
	Byte Count	D0	0	24-31(*)
		C1	0	0-7
			1	8-15
	Requester Address	D1	0	16-23
		98	0	0-7
			1	8-15
		99	0	16-23
			1	24-31(*)
Channel 5	Target Address	C2	0	0-7
			1	8-15
		8B	x	16-23
	Byte Count	D2	0	24-31(*)
		C3	0	0-7
			1	8-15
	Requester Address	D3	0	16-23
		9A	0	0-7
			1	8-15
		9B	0	16-23
			1	24-31(*)
Channel 6	Target Address	C4	0	0-7
			1	8-15
		89	x	16-23
	Byte Count	D4	0	24-31(*)
		C5	0	0-7
			1	8-15
	Requester Address	D5	0	16-23
		9C	0	0-7
			1	8-15
		9D	0	16-23
			1	24-31(*)
Channel 7	Target Address	C6	0	0-7
			1	8-15
		8A	x	16-23
	Byte Count	D6	0	24-31(*)
		C7	0	0-7
			1	8-15
	Requester Address	D7	0	16-23
		9E	0	0-7
			1	8-15
		9F	0	16-23
			1	24-31(*)

NOTE:

(*)These bits are not available externally. You need to be aware of their existence for chaining and Byte Pointer Flip-Flop operations. Please see section 3.5.2 for further details.

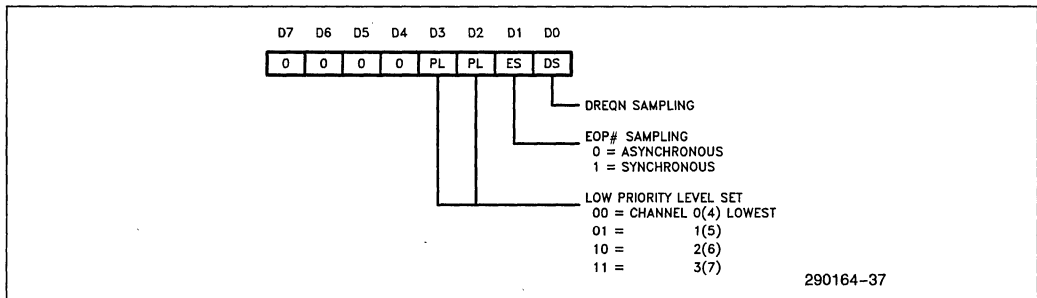
Command Register I (write only)

Port Addresses— Channels 0–3—0008H
 Channels 4–7—00C8H



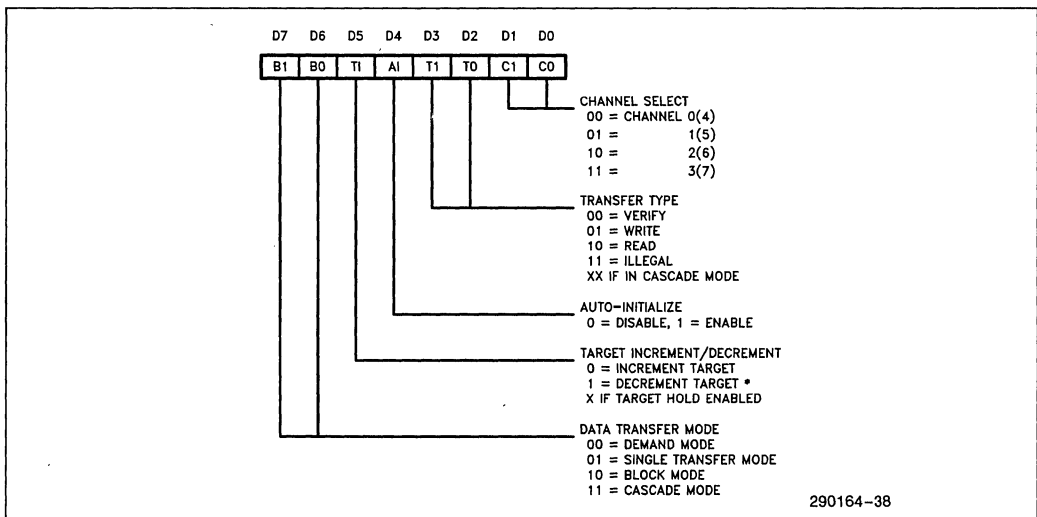
Command Register II (write only)

Port Addresses— Channels 0–3—001AH
 Channels 4–7—00DAH



Mode Register I (write only)

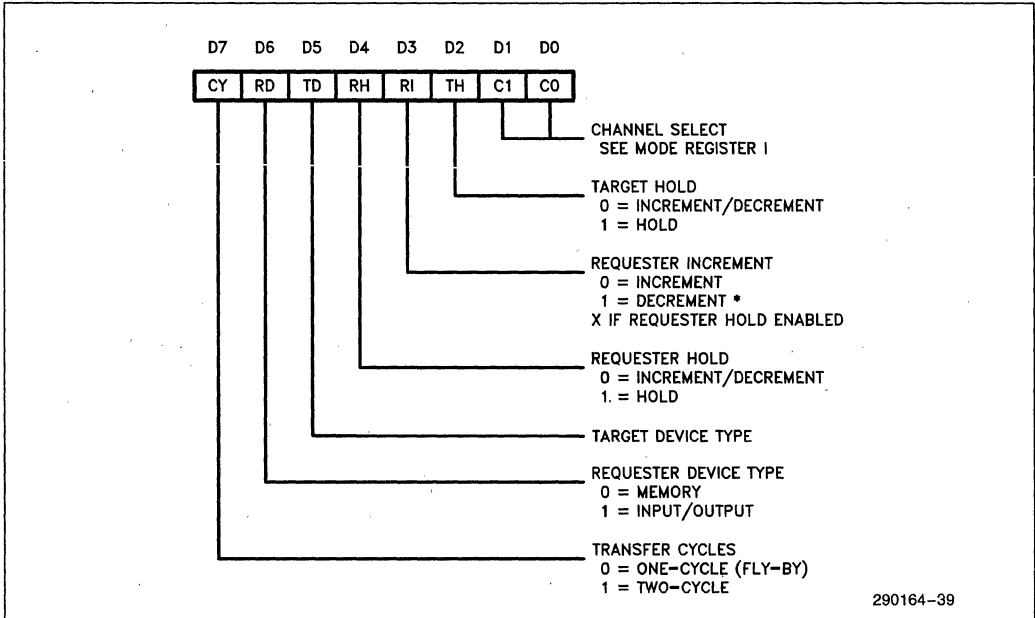
Port Addresses— Channels 0–3—000BH
 Channels 4–7—00CBH



*Target and Requester DECREMENT is allowed only for byte transfers.

Mode Register II (write only)

Port Addresses— Channels 0–3—001BH
 Channels 4–7—00DBH

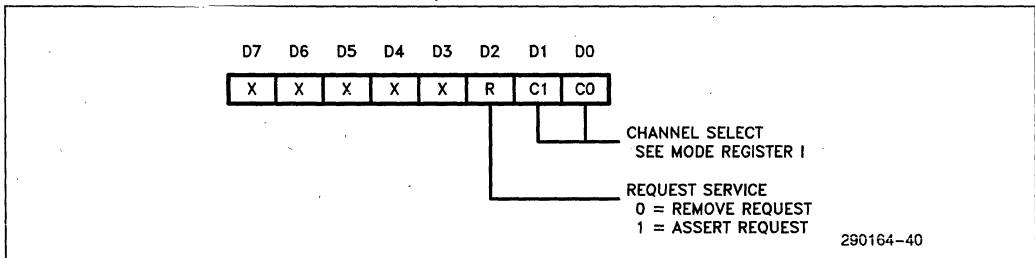


*Target and Requester DECREMENT is allowed only for byte transfers.

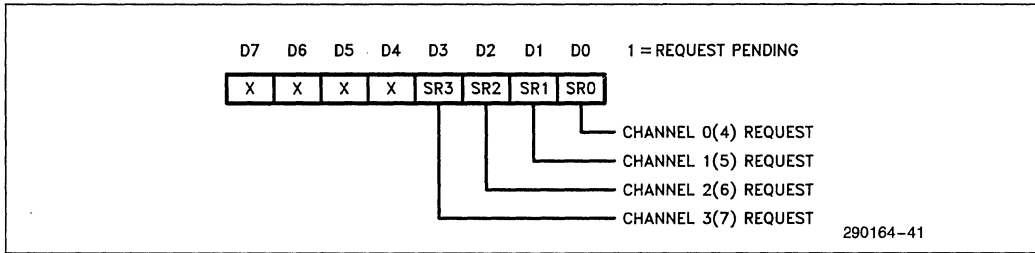
Software Request Register (read/write)

Port Addresses— Channels 0–3—0009H
 Channels 4–7—00C9H

Write Format: Software DMA Service Request

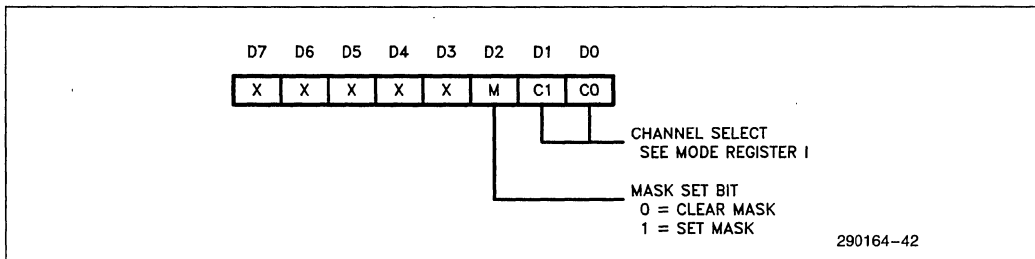


Read Format: Software Requests Pending



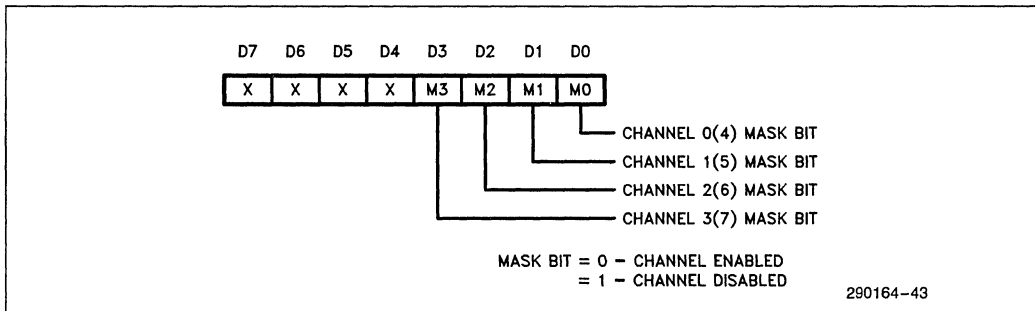
Mask Set/Reset Register Individual Channel Mask (write only)

Port Addresses— Channels 0-3—000AH
Channels 4-7—00CAH



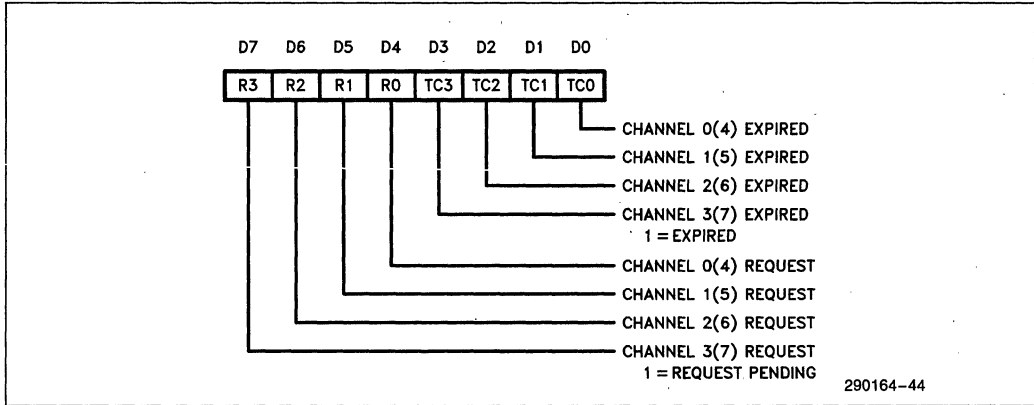
Mask Read/Write Register Group Channel Mask (read/write)

Port Addresses— Channels 0-3—000FH
Channels 4-7—00CFH



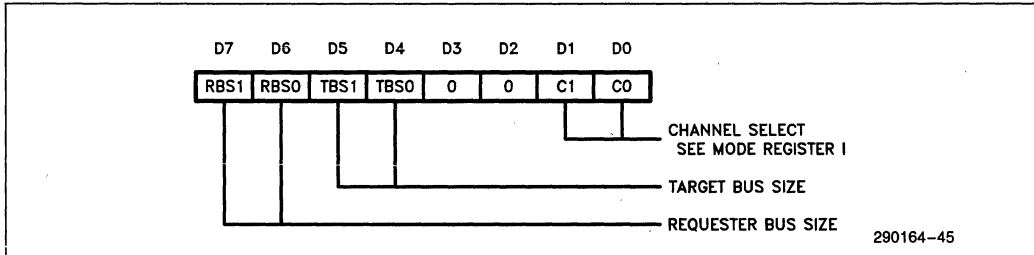
Status Register Channel Process Status (read only)

Port Addresses— Channels 0–3—0008H
 Channels 4–7—00C8H



Bus Size Register Set Data Path Width (write only)

Port Addresses— Channels 0–3—0018H
 Channels 4–7—00D8H



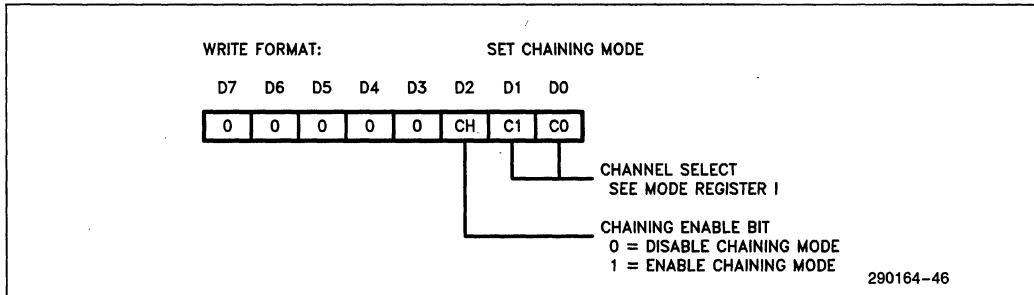
Bus Size Encoding:

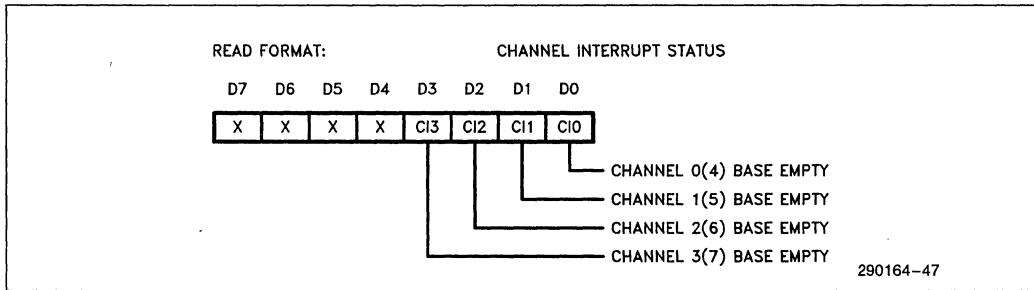
- 00 = Reserved by intel 10 = 16-bit Bus
- 01 = 32-bit Bus* 11 = 8-bit Bus

*If programmed as 32-bit bus width, the corresponding device will be accessed in two 16-bit cycles provided that the data is aligned within word boundary.

Chaining Register (read/write)

Port Addresses— Channels 0–3—0019H
 Channels 4–7—00D9H





3.8 8237A Compatibility

The register arrangement of the 82370 DMA Controller is a superset of the 8237A DMA Controller. Functionally the 82370 DMA Controller is very different from the 8237A. Most of the functions of the 8237A are performed also by the 82370. The following discussion points out the differences between the 8237A and the 82370.

The 8237A is limited to transfers between I/O and memory only (except in one special case, where two channels can be used to perform memory-to-memory transfers). The 82370 DMA Controller can transfer between any combination of memory and I/O. Several other features of the 8237A are enhanced or expanded in the 82370 and other features are added.

The 8237A is an 8-bit only DMA device. For programming compatibility, all of the 8-bit registers are preserved in the 82370. The 82370 is programmed via 8-bit registers. The address registers in the 82370 are 24-bit registers in order to support the 80376's 24-bit bus. The Byte Count Registers are 24-bit registers, allowing support of larger data blocks than possible with the 8237A.

All of the 8237A's operating modes are supported by the 82370 (except the cumbersome two-channel memory-to-memory transfer). The 82370 performs memory-to-memory transfers using only one channel. The 82370 has the added features of buffer pipelining (Buffer Chaining Process) and programmable priority levels.

The 82370 also adds the feature of address registers for both destination and source. These addresses may be incremented, decremented, or held constant, as required by the application of the individual channel. This allows any combination of destination and source device.

Each DMA channel has associated with it a Target and a Requester. In the 8237A, the Target is the device which can be accessed by the address register, the Requester is the device which is accessed by the DMA Acknowledge signals and must be an I/O device.

4.0 PROGRAMMABLE INTERRUPT CONTROLLER (PIC)

4.1 Functional Description

The 82370 Programmable Interrupt Controller (PIC) consists of three enhanced 82C59A Interrupt Controllers. These three controllers together provide 15 external and 5 internal interrupt request inputs. Each external request input can be cascaded with an additional 82C59A slave controller. This scheme allows the 82370 to support a maximum of 120 (15 x 8) external interrupt request inputs.

Following one or more interrupt requests, the 82370 PIC issues an interrupt signal to the 80376. When the 80376 host processor responds with an interrupt acknowledge signal, the PIC will arbitrate between the pending interrupt requests and place the interrupt vector associated with the highest priority pending request on the data bus.

The major enhancement in the 82370 PIC over the 82C59A is that each of the interrupt request inputs can be individually programmed with its own interrupt vector, allowing more flexibility in interrupt vector mapping.

4.1.1 INTERNAL BLOCK DIAGRAM

The block diagram of the 82370 Programmable Interrupt Controller is shown in Figure 4-1. Internally,

the PIC consists of three 82C59A banks: A, B and C. The three banks are cascaded to one another: C is cascaded to B, B is cascaded to A. The INT output of Bank A is used externally to interrupt the 80376.

Bank A has nine interrupt request inputs (two are unused), and Banks B and C have eight interrupt request inputs. Of the fifteen external interrupt request inputs, two are shared by other functions. Specifically, the Interrupt Request 3 input (IRQ3#) can be used as the Timer 2 output (TOUT2#). This pin can be used in three different ways: IRQ3# input only, TOUT2# output only, or using TOUT2# to generate an IRQ3# interrupt request. Also, the Interrupt Request 9 input (IRQ9#) can be used as DMA Request 4 input (DREQ 4). Typically, only IRQ9# or DREQ4 can be used at a time.

4.1.2 INTERRUPT CONTROLLER BANKS

All three banks are identical, with the exception of the IRQ1.5 on Bank A. Therefore, only one bank will be discussed. In the 82370 PIC, all external requests can be cascaded into and each interrupt controller bank behaves like a master. As compared to the 82C59A, the enhancements in the banks are:

- All interrupt vectors are individually programmable. (In the 82C59A, the vectors must be programmed in eight consecutive interrupt vector locations.)
- The cascade address is provided on the Data Bus (D0-D7). (In the 82C59A, three dedicated control signals (CAS0, CAS1, CAS2) are used for master/slave cascading.)

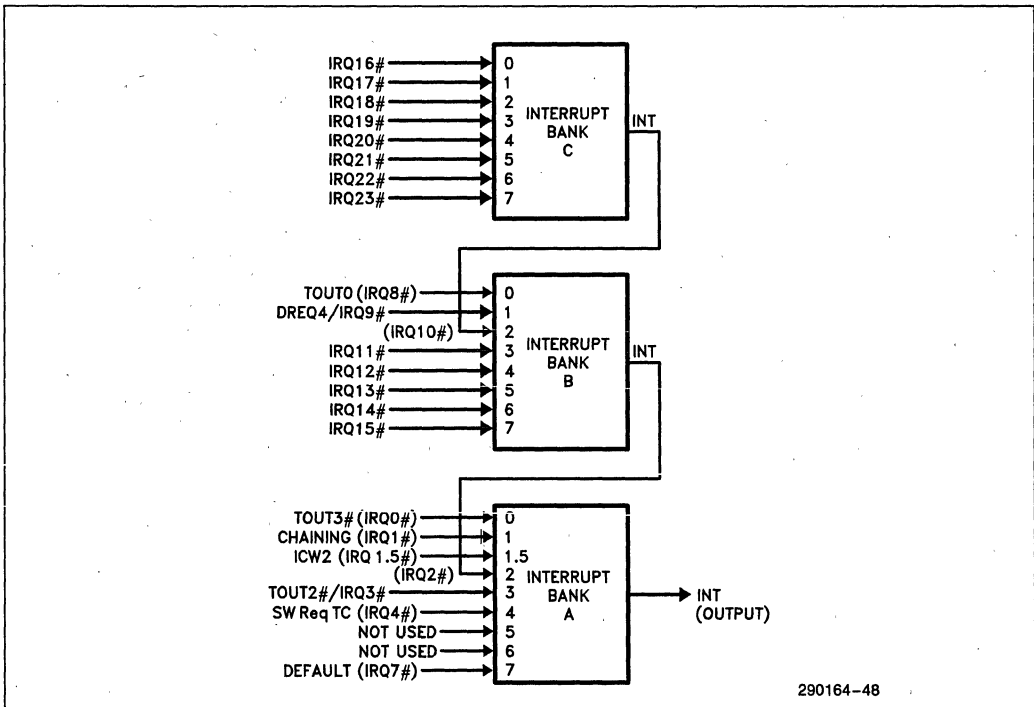


Figure 4-1. Interrupt Controller Block Diagram

290164-48

The block diagram of a bank is shown in Figure 4-2. As can be seen from this figure, the bank consists of six major blocks: the Interrupt Request Register (IRR), the In-Service Register (ISR), the Interrupt Mask Register (IMR), the Priority Resolver (PR), the Vector Registers (VR), and the Control Logic. The functional description of each block is included below.

INTERRUPT REQUEST (IRR) AND IN-SERVICE REGISTER (ISR)

The interrupts at the Interrupt Request (IRQ) input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service Register (ISR). The IRR is used to store all interrupt levels which are requesting service; and the ISR is used to store all interrupt levels which are being serviced.

PRIORITY RESOLVER (PR)

This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during an Interrupt Acknowledge cycle.

INTERRUPT MASK REGISTER (IMR)

The IMR stores the bits which mask the interrupt lines to be masked (disabled). The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower priority.

VECTOR REGISTERS (VR)

This block contains a set of Vector Registers, one for each interrupt request line, to store the pre-programmed interrupt vector number. The corresponding vector number will be driven onto the Data Bus of the 82370 during the Interrupt Acknowledge cycle.

CONTROL LOGIC

The Control Logic coordinates the overall operations of the other internal blocks within the same bank. This logic will drive the Interrupt Output signal (INT) HIGH when one or more unmasked interrupt inputs are active (LOW). The INT output signal goes directly to the 80376 (in bank A) or to another bank to which this bank is cascaded (see Figure 4-1). Also,

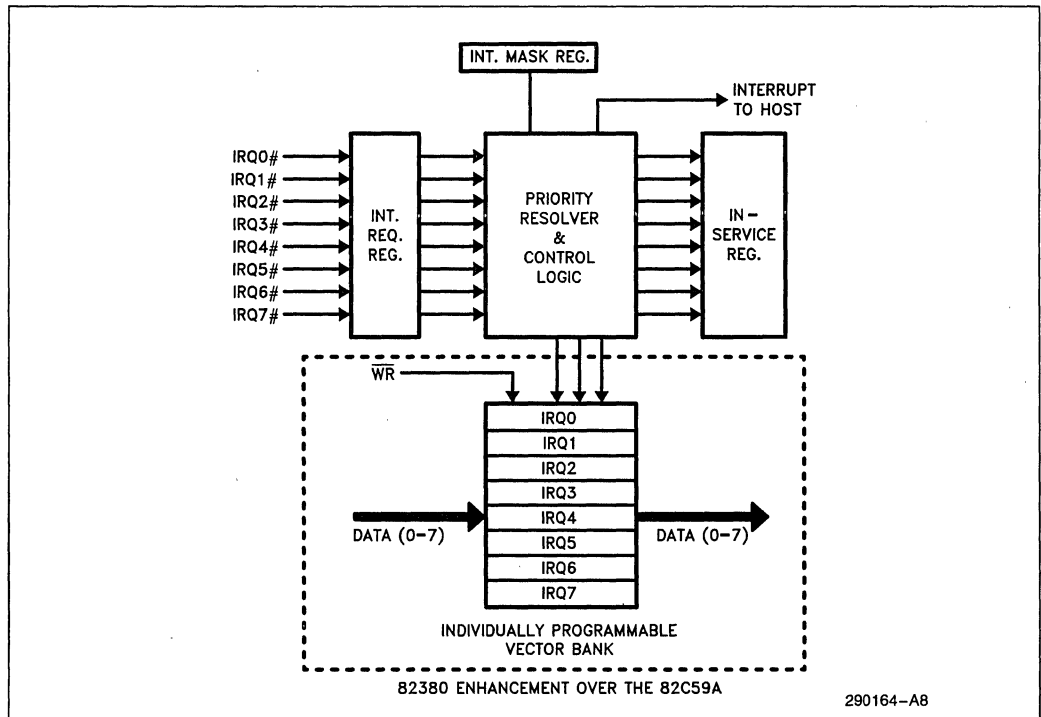


Figure 4-2. Interrupt Bank Block Diagram

this logic will recognize an Interrupt Acknowledge cycle (via M/IO#, D/C# and W/R# signals). During this bus cycle, the Control Logic will enable the corresponding Vector Register to drive the interrupt vector onto the Data Bus.

In bank A, the Control Logic is also responsible for handling the special ICW2 interrupt request input (IRQ1.5).

4.2 Interface Signals

4.2.1 INTERRUPT INPUTS

There are 15 external Interrupt Request inputs and 5 internal Interrupt Requests. The external request inputs are: IRQ3#, IRQ9#, IRQ11# to IRQ23#. They are shown in bold arrows in Figure 4-1. All IRQ inputs are active LOW and they can be programmed (via a control bit in the Initialization Command Word 1 (ICW1)) to be either edge-triggered or level-triggered. In order to be recognized as a valid interrupt request, the interrupt input must be active (LOW) until the first INTA cycle (see Bus Functional Description). Note that all 15 external Interrupt Request inputs have weak internal pull-up resistors.

As mentioned earlier, an 82C59A can be cascaded to each external interrupt input to expand the interrupt capacity to a maximum of 120 levels. Also, two of the interrupt inputs are dual functions: IRQ3# can be used as Timer 2 output (TOUT2#) and IRQ9# can be used as DREQ4 input. IRQ3# is a bidirectional dual function pin. This interrupt request input is wired-OR with the output of Timer 2 (TOUT2#). If only IRQ3# function is to be used, Timer 2 should be programmed so that OUT2 is LOW. Note that TOUT2# can also be used to generate an interrupt request to IRQ3# input.

The five internal interrupt requests serve special system functions. They are shown in Table 4-1. The following paragraphs describe these interrupts.

Table 4-1. 82370 Internal Interrupt Requests

Interrupt Request	Interrupt Source
IRQ0#	Timer 3 Output (TOUT3)
IRQ8#	Timer 0 Output (TOUT0)
IRQ1#	DMA Chaining Request
IRQ4#	DMA Terminal Count
IRQ1.5#	ICW2 Written

TIMER 0 AND TIMER 3 INTERRUPT REQUESTS

IRQ8# and IRQ0# interrupt requests are initiated by the output of Timers 0 and 3, respectively. Each of these requests is generated by an edge-detector flip-flop.

The flip-flops are activated by the following conditions:

- Set — Rising edge of timer output (TOUT);
- Clear — Interrupt acknowledge for this request; OR Request is masked (disabled); OR Hardware Reset.

CHAINING AND TERMINAL COUNT INTERRUPTS

These interrupt requests are generated by the 82370 DMA Controller. The chaining request (IRQ1#) indicates that the DMA Base Register is not loaded. The Terminal Count request (IRQ4#) indicates that a software DMA request was cleared.

ICW2 INTERRUPT REQUEST

Whenever an Initialization Control Word 2 (ICW2) is written to a Bank, a special ICW2 interrupt request is generated. The interrupt will be cleared when the newly programmed ICW2 Register is read. This interrupt request is in Bank A at level 1.5. This interrupt request is internally ORed with the Cascaded Request from Bank B and is always assigned a higher priority than the Cascaded Request.

This special interrupt is provided to support compatibility with the original 82C59A. A detailed description of this interrupt is discussed in the Programming section.

DEFAULT INTERRUPT (IRQ7#)

During an Interrupt Acknowledge cycle, if there is no active pending request, the PIC will automatically generate a default vector. This vector corresponds to the IRQ7# vector in bank A.

4.2.2 INTERRUPT OUTPUT (INT)

The INT output pin is taken directly from bank A. This signal should be tied to the Maskable Interrupt Request (INTR) of the 80376. When this signal is active (HIGH), it indicates that one or more internal/external interrupt requests are pending. The 80376 is expected to respond with an interrupt acknowledge cycle.

4.3 Bus Functional Description

The INT output of bank A will be activated as a result of any unmasked interrupt request. This may be a non-cascaded or cascaded request. After the PIC has driven the INT signal HIGH, the 80376 will respond by performing two interrupt acknowledge cycles. The timing diagram in Figure 4-3 shows a typical interrupt acknowledge process between the 82370 and the 80376 CPU.

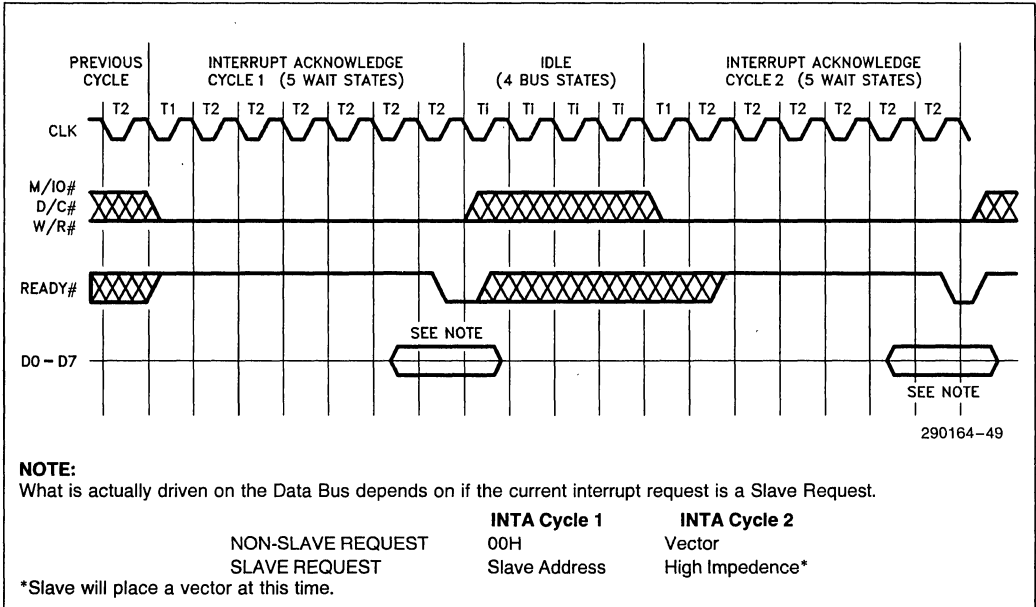


Figure 4-3. Interrupt Acknowledge Cycle

After activating the INT signal, the 82370 monitors the status lines (M/IO#, D/C#, W/R#) and waits for the 80376 to initiate the first interrupt acknowledge cycle. In the 80376 environment, two successive interrupt acknowledge cycles (INTA) marked by M/IO# = LOW, D/C# = LOW, and W/R# = LOW are performed. During the first INTA cycle, the PIC will determine the highest priority request. Assuming this interrupt input has no external Slave Controller cascaded to it, the 82370 will drive the Data Bus with 00H in the first INTA cycle. During the second INTA cycle, the 82370 PIC will drive the Data Bus with the corresponding pre-programmed interrupt vector.

If the PIC determines (from the ICW3) that this interrupt input has an external Slave Controller cascaded to it, it will drive the Data Bus with the specific Slave Cascade Address (instead of 00H) during the first INTA cycle. This Slave Cascade Address is the pre-programmed content in the corresponding Vector Register. This means that no Slave Address should be chosen to be 00H. Note that the Slave Address and Interrupt Vector are different interpretations of the same thing. They are both the contents of the programmable Vector Register. During the second INTA cycle, the Data Bus will be floated so that the external Slave Controller can drive its interrupt vector on the bus. Since the Slave Interrupt Controller resides on the system bus, bus transceiver enable and direction control logic must take this into consideration.

In order to have a successful interrupt service, the interrupt request input must be held valid (LOW) until the beginning of the first interrupt acknowledge cycle. If there is no pending interrupt request when the first INTA cycle is generated, the PIC will generate a default vector, which is the IRQ7 vector (Bank A, level 7).

According to the Bus Cycle definition of the 80376, there will be four Bus Idle States between the two interrupt acknowledge cycles. These idle bus cycles will be initiated by the 80376. Also, during each interrupt acknowledge cycle, the internal Wait State Generator of the 82370 will automatically generate the required number of wait states for internal delays.

4.4 Modes of Operation

A variety of modes and commands are available for controlling the 82370 PIC. All of them are programmable; that is, they may be changed dynamically under software control. In fact, each bank can be programmed individually to operate in different modes. With these modes and commands, many possible configurations are conceivable, giving the user enough versatility for almost any interrupt controlled application.

This section is not intended to show how the 82370 PIC can be programmed. Rather, it describes the operation in different modes.

4.4.1 END-OF-INTERRUPT

Upon completion of an interrupt service routine, the interrupted bank needs to be notified so its ISR can be updated. This allows the PIC to keep track of which interrupt levels are in the process of being serviced and their relative priorities. Three different End-Of-Interrupt (EOI) formats are available. They are: Non-Specific EOI Command, Specific EOI Command, and Automatic EOI Mode. Selection of which EOI to use is dependent upon the interrupt operations the user wishes to perform.

If the 82370 is NOT programmed in the Automatic EOI Mode, an EOI command must be issued by the 80376 to the specific 82370 PIC Controller Bank. Also, if this controller bank is cascaded to another internal bank, an EOI command must also be sent to the bank to which this bank is cascaded. For example, if an interrupt request of Bank C in the 82370 PIC is serviced, an EOI should be written into Bank C, Bank B and Bank A. If the request comes from an external interrupt controller cascaded to Bank C, then an EOI should be written into the external controller as well.

NON-SPECIFIC EOI COMMAND

A Non-Specific EOI command sent from the 80376 lets the 82370 PIC bank know when a service routine has been completed, without specification of its exact interrupt level. The respective interrupt bank automatically determines the interrupt level and resets the correct bit in the ISR.

To take advantage of the Non-Specific EOI, the interrupt bank must be in a mode of operation in which it can predetermine its in-service routine levels. For this reason, the Non-Specific EOI command should only be used when the most recent level acknowledged and serviced is always the highest priority level (i.e. in the Fully Nested Mode structure to be described below). When the interrupt bank receives a Non-Specific EOI command, it simply resets the highest priority ISR bit to indicate that the highest priority routine in service is finished.

Special consideration should be taken when deciding to use the Non-Specific EOI command. Here are two operating conditions in which it is best NOT used since the Fully Nested Mode structure will be destroyed:

- Using the Set Priority command within an interrupt service routine.
- Using a Special Mask Mode.

These conditions are covered in more detail in their own sections, but are listed here for reference.

SPECIFIC EOI COMMAND

Unlike a Non-Specific EOI command which automatically resets the highest priority ISR bit, a Specific EOI command specifies an exact ISR bit to be reset. Any one of the IRQ levels of an interrupt bank can be specified in the command.

The Specific EOI command is needed to reset the ISR bit of a completed service routine whenever the interrupt bank is not able to automatically determine it. The Specific EOI command can be used in all conditions of operation, including those that prohibit Non-Specific EOI command usage mentioned above.

AUTOMATIC EOI MODE

When programmed in the Automatic EOI Mode, the 80376 no longer needs to issue a command to notify the interrupt bank it has completed an interrupt routine. The interrupt bank accomplishes this by performing a Non-Specific EOI automatically at the end of the second INTA cycle.

Special consideration should be taken when deciding to use the Automatic EOI Mode because it may disturb the Fully Nested Mode structure. In the Automatic EOI Mode, the ISR bit of a routine in service is reset right after it is acknowledged, thus leaving no designation in the ISR that a service routine is being executed. If any interrupt request within the same bank occurs during this time and interrupts are enabled, it will get serviced regardless of its priority. Therefore, when using this mode, the 80376 should keep its interrupt request input disabled during execution of a service routine. By doing this, higher priority interrupt levels will be serviced only after the completion of a routine in service. This guideline restores the Fully Nested Mode structure. However, in this scheme, a routine in service cannot be interrupted since the host's interrupt request input is disabled.

4.4.2 INTERRUPT PRIORITIES

The 82370 PIC provides various methods for arranging the interrupt priorities of the interrupt request inputs to suit different applications. The following subsections explain these methods in detail.

4.4.2.1 Fully Nested Mode

The Fully Nested Mode of operation is a general purpose priority mode. This mode supports a multi-level interrupt structure in which all of the Interrupt Request (IRQ) inputs within one bank are arranged from highest to lowest.

Unless otherwise programmed, the Fully Nested Mode is entered by default upon initialization. At this time, IRQ0# is assigned the highest priority (priority=0) and IRQ7# the lowest (priority=7). This default priority can be changed, as will be explained later in the Rotating Priority Mode.

When an interrupt is acknowledged, the highest priority request is determined from the Interrupt Request Register (IRR) and its vector is placed on the bus. In addition, the corresponding bit in the In-Service Register (ISR) is set to designate the routine in service. This ISR bit will remain set until the 80376 issues an End Of Interrupt (EOI) command immediately before returning from the service routine; or alternately, if the Automatic End Of Interrupt (AEOI) bit is set, the ISR bit will be reset at the end of the second INTA cycle.

While the ISR bit is set, all further interrupts of the same or lower priority are inhibited. Higher level interrupts can still generate an interrupt, which will be acknowledged only if the 80376 internal interrupt enable flip-flop has been reenabled (through software inside the current service routine).

4.4.2.2 Automatic Rotation–Equal Priority Devices

Automatic rotation of priorities serves in applications where the interrupting devices are of equal priority

within an interrupt bank. In this kind of environment, once a device is serviced, all other equal priority peripherals should be given a chance to be serviced before the original device is serviced again. This is accomplished by automatically assigning a device the lowest priority after being serviced. Thus, in the worst case, the device would have to wait until all other peripherals connected to the same bank are serviced before it is serviced again.

There are two methods of accomplishing automatic rotation. One is used in conjunction with the Non-Specific EOI command and the other is used with the Automatic EOI mode. These two methods are discussed below.

ROTATE ON NON-SPECIFIC EOI COMMAND

When the Rotate On Non-Specific EOI command is issued, the highest ISR bit is reset as in a normal Non-Specific EOI command. However, after it is reset, the corresponding Interrupt Request (IRQ) level is assigned the lowest priority. Other IRQ priorities rotate to conform to the Fully Nested Mode based on the newly assigned low priority.

Figure 4-4 shows how the Rotate On Non-Specific EOI command affects the interrupt priorities. Assume the IRQ priorities were assigned with IRQ0 the highest and IRQ7 the lowest. IRQ6 and IRQ4 are

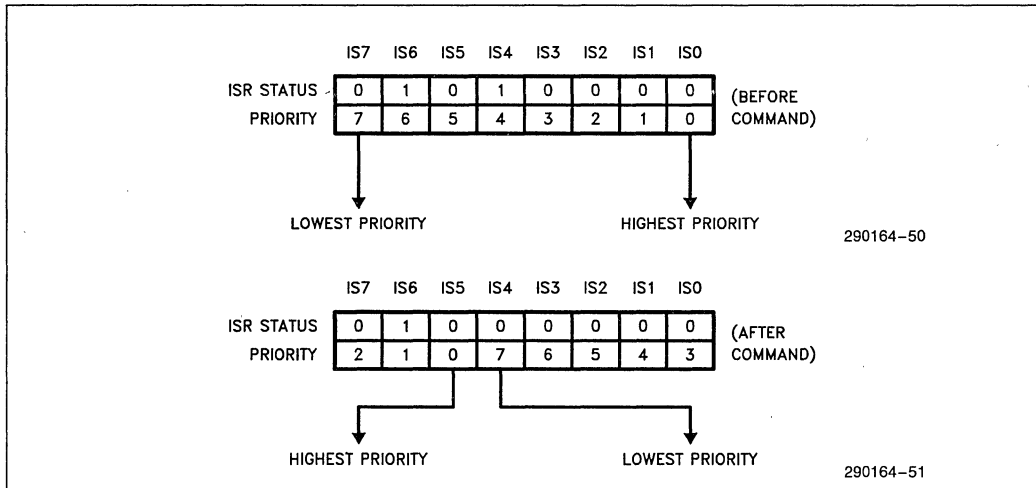


Figure 4-4. Rotate On Non-Specific EOI Command

already in service but neither is completed. Being the higher priority routine, IRQ4 is necessarily the routine being executed. During the IRQ4 routine, a rotate on Non-Specific EOI command is executed. When this happens, Bit 4 in the ISR is reset. IRQ4 then becomes the lowest priority and IRQ5 becomes the highest.

ROTATE ON AUTOMATIC EOI MODE

The Rotate On Automatic EOI Mode works much like the Rotate On Non-Specific EOI Command. The main difference is that priority rotation is done automatically after the second INTA cycle of an interrupt request. To enter or exit this mode, a Rotate-On-Automatic-EOI Set Command and Rotate-On-Automatic-EOI Clear Command is provided. After this mode is entered, no other commands are needed as in the normal Automatic EOI Mode. However, it must be noted again that when using any form of the Automatic EOI Mode, special consideration should be taken. The guideline presented in the Automatic EOI Mode also applies here.

4.4.2.3 Specific Rotation-Specific Priority

Specific rotation gives the user versatile capabilities in interrupt controlled operations. It serves in those applications in which a specific device's interrupt priority must be altered. As opposed to Automatic Rotation which will automatically set priorities after each interrupt request is serviced, specific rotation is completely user controlled. That is, the user selects which interrupt level is to receive the lowest or the highest priority. This can be done during the main program or within interrupt routines. Two specific ro-

tation commands are available to the user: Set Priority Command and Rotate On Specific EOI Command.

SET PRIORITY COMMAND

The Set Priority Command allows the programmer to assign an IRQ level the lowest priority. All other interrupt levels will conform to the Fully Nested Mode based on the newly assigned low priority.

ROTATE ON SPECIFIC EOI COMMAND

The Rotate On Specific EOI Command is literally a combination of the Set Priority Command and the Specific EOI Command. Like the Set Priority Command, a specified IRQ level is assigned lowest priority. Like the Specific EOI Command, a specified level will be reset in the ISR. Thus, this command accomplishes both tasks in one single command.

4.4.2.4 Interrupt Priority Mode Summary

In order to simplify understanding the many modes of interrupt priority, Table 4-2 is provided to bring out their summary of operations.

4.4.3 INTERRUPT MASKING

VIA INTERRUPT MASK REGISTER

Each bank in the 82370 PIC has an Interrupt Mask Register (IMR) which enhances interrupt control ca-

Table 4-2. Interrupt Priority Mode Summary

Interrupt Priority Mode	Operation Summary	Effect On Priority After EOI	
		Non-Specific/Automatic	Specific
Fully-Nested Mode	IRQ0 # - Highest Priority IRQ7 # - Lowest Priority	No change in priority. Highest ISR bit is reset.	Not Applicable.
Automatic Rotation (Equal Priority Devices)	Interrupt level just serviced is the lowest priority. Other priorities rotate to conform to Fully-Nested Mode.	Highest ISR bit is reset and the corresponding level becomes the lowest priority.	Not Applicable.
Specific Rotation (Specific Priority Devices)	User specifies the lowest priority level. Other priorities rotate to conform to Fully-Nested Mode.	Not Applicable.	As described under "Operation Summary".

pabilities. This IMR allows individual IRQ masking. When an IRQ is masked, its interrupt request is disabled until it is unmasked. Each bit in the 8-bit IMR disables one interrupt channel if it is set (HIGH). Bit 0 masks IRQ0, Bit 1 masks IRQ1 and so forth. Masking an IRQ channel will only disable the corresponding channel and does not affect the others' operations.

The IMR acts only on the output of the IRR. That is, if an interrupt occurs while its IMR bit is set, this request is not "forgotten". Even with an IRQ input masked, it is still possible to set the IRR. Therefore, when the IMR bit is reset, an interrupt request to the 80376 will then be generated, providing that the IRQ request remains active. If the IRQ request is removed before the IMR is reset, the Default Interrupt Vector (Bank A, level 7) will be generated during the interrupt acknowledge cycle.

SPECIAL MASK MODE

In the Fully Nested Mode, all IRQ levels of lower priority than the routine in service are inhibited. However, in some applications, it may be desirable to let a lower priority interrupt request to interrupt the routine in service. One method to achieve this is by using the Special Mask Mode. Working in conjunction with the IMR, the Special Mask Mode enables interrupts from all levels except the level in service. This is usually done inside an interrupt service routine by masking the level that is in service and then issuing the Special Mask Mode Command. Once the Special Mask Mode is enabled, it remains in effect until it is disabled.

4.4.4 EDGE OR LEVEL INTERRUPT TRIGGERING

Each bank in the 82370 PIC can be programmed independently for either edge or level sensing for the

interrupt request signals. Recall that all IRQ inputs are active LOW. Therefore, in the edge triggered mode, an active edge is defined as an input transition from an inactive (HIGH) to active (LOW) state. The interrupt input may remain active without generating another interrupt. During level triggered mode, an interrupt request will be recognized by an active (LOW) input, and there is no need for edge detection. However, the interrupt request must be removed before the EOI Command is issued, or the 80376 must be disabled to prevent a second false interrupt from occurring.

In either modes, the interrupt request input must be active (LOW) during the first INTA cycle in order to be recognized. Otherwise, the Default Interrupt Vector will be generated at level 7 of Bank A.

4.4.5 INTERRUPT CASCADING

As mentioned previously, the 82370 allows for external Slave interrupt controllers to be cascaded to any of its external interrupt request pins. The 82370 PIC indicates that an external Slave Controller is to be serviced by putting the contents of the Vector Register associated with the particular request on the 80376 Data Bus during the first INTA cycle (instead of 00H during a non-slave service). The external logic should latch the vector on the Data Bus using the INTA status signals and use it to select the external Slave Controller to be serviced (see Figure 4-5). The selected Slave will then respond to the second INTA cycle and place its vector on the Data Bus. This method requires that if external Slave Controllers are used in the system, no vector should be programmed to 00H.

Since the external Slave Cascade Address is provided on the Data Bus during INTA cycle 1, an external latch is required to capture this address for the Slave Controller. A simple scheme is depicted in Figure 4-5 below.

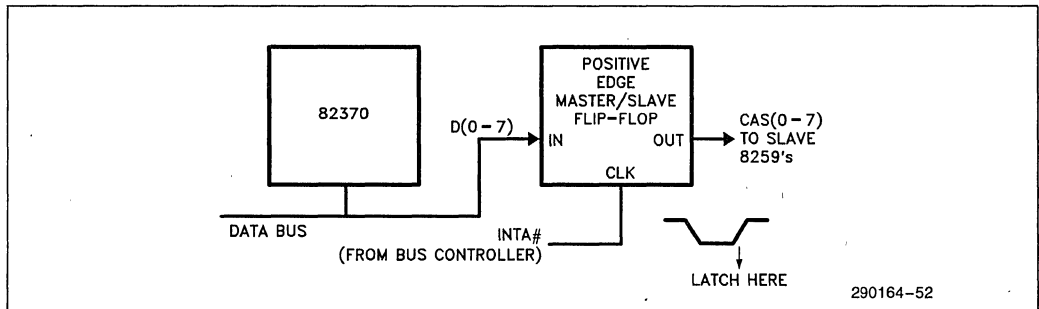


Figure 4-5. Slave Cascade Address Capturing

4.4.5.1 Special Fully Nested Mode

This mode will be used where cascading is employed and the priority is to be conserved within each Slave Controller. The Special Fully Nested Mode is similar to the "regular" Fully Nested Mode with the following exceptions:

- When an interrupt request from a Slave Controller is in service, this Slave Controller is not locked out from the Master's priority logic. Further interrupt requests from the higher priority logic within the Slave Controller will be recognized by the 82370 PIC and will initiate interrupts to the 80376. In comparing to the "regular" Fully Nested Mode, the Slave Controller is masked out when its request is in service and no higher requests from the same Slave Controller can be serviced.
- Before exiting the interrupt service routine, the software has to check whether the interrupt serviced was the only request from the Slave Controller. This is done by sending a Non-Specific EOI Command to the Slave Controller and then reading its In Service Register. If there are no requests in the Slave Controller, a Non-Specific EOI can be sent to the corresponding 82370 PIC bank also. Otherwise, no EOI should be sent.

4.4.6 READING INTERRUPT STATUS

The 82370 PIC provides several ways to read different status of each interrupt bank for more flexible interrupt control operations. These include polling the highest priority pending interrupt request and reading the contents of different interrupt status registers.

4.4.6.1 Poll Command

The 82370 PIC supports status polling operations with the Poll Command. In a Poll Command, the pending interrupt request with the highest priority can be determined. To use this command, the INT output is not used, or the 80376 interrupt is disabled. Service to devices is achieved by software using the Poll Command.

This mode is useful if there is a routine command common to several levels so that the INTA sequence is not needed. Another application is to use the Poll Command to expand the number of priority levels.

Notice that the ICW2 mechanism is not supported for the Poll Command. However, if the Poll Command is used, the programmable Vector Registers are of no concern since no INTA cycle will be generated.

4.4.6.2 Reading Interrupt Registers

The contents of each interrupt register (IRR, ISR, and IMR) can be read to update the user's program on the present status of the 82370 PIC. This can be a versatile tool in the decision making process of a service routine, giving the user more control over interrupt operations.

The reading of the IRR and ISR contents can be performed via the Operation Control Word 3 by using a Read Status Register Command and the content of IMR can be read via a simple read operation of the register itself.

4.5 Register Set Overview

Each bank of the 82370 PIC consists of a set of 8-bit registers to control its operations. The address map of all the registers is shown in Table 4-3 below. Since all three register sets are identical in functions, only one set will be described.

Functionally, each register set can be divided into five groups. They are: the four Initialization Command Words (ICW's), the three Operation Control Words (OCW's), the Poll/interrupt Request/in-Service Register, the Interrupt Mask Register, and the Vector Registers. A description of each group follows.

Table 4-3. Interrupt Controller Register Address Map

Port Address	Access	Register Description
20H	Write Read	Bank B ICW1, OCW2, or OCW3 Bank B Poll, Request or In-Service Status Register
21H	Write Read	Bank B ICW2, ICW3, ICW4, OCW1 Bank B Mask Register
22H	Read	Bank B ICW2
28H	Read/Write	IRQ8 Vector Register
29H	Read/Write	IRQ9 Vector Register
2AH	Read/Write	Reserved
2BH	Read/Write	IRQ11 Vector Register
2CH	Read/Write	IRQ12 Vector Register
2DH	Read/Write	IRQ13 Vector Register
2EH	Read/Write	IRQ14 Vector Register
2FH	Read/Write	IRQ15 Vector Register
A0H	Write Read	Bank C ICW1, OCW2, or OCW3 Bank C Poll, Request or In-Service Status Register
A1H	Write Read	Bank C ICW2, ICW3, ICW4, OCW1 Bank C Mask Register
A2H	Read	Bank C ICW2
A8H	Read/Write	IRQ16 Vector Register
A9H	Read/Write	IRQ17 Vector Register
AAH	Read/Write	IRQ18 Vector Register
ABH	Read/Write	IRQ19 Vector Register
ACH	Read/Write	IRQ20 Vector Register
ADH	Read/Write	IRQ21 Vector Register
AEH	Read/Write	IRQ22 Vector Register
AFH	Read/Write	IRQ23 Vector Register
30H	Write Read	Bank A ICW1, OCW2, or OCW3 Bank A Poll, Request or In-Service Status Register
31H	Write Read	Bank A ICW2, ICW3, ICW4, OCW1 Bank A Mask Register
32H	Read	Bank ICW2
38H	Read/Write	IRQ0 Vector Register
39H	Read/Write	IRQ1 Vector Register
3AH	Read/Write	IRQ1.5 Vector Register
3BH	Read/Write	IRQ3 Vector Register
3CH	Read/Write	IRQ4 Vector Register
3DH	Read/Write	Reserved
3EH	Read/Write	Reserved
3FH	Read/Write	IRQ7 Vector Register

4.5.1 INITIALIZATION COMMAND WORDS (ICW)

Before normal operation can begin, the 82370 PIC must be brought to a known state. There are four 8-bit Initialization Command Words in each interrupt bank to setup the necessary conditions and modes for proper operation. Except for the second command word (ICW2) which is a read/write register, the other three are write-only registers. Without going into detail of the bit definitions of the command words, the following subsections give a brief description of what functions each command word controls.

ICW1

The ICW1 has three major functions. They are:

- To select between the two IRQ input triggering modes (edge- or level-triggered);
- To designate whether or not the interrupt bank is to be used alone or in the cascade mode. If the cascade mode is desired, the interrupt bank will accept ICW3 for further cascade mode programming. Otherwise, no ICW3 will be accepted;
- To determine whether or not ICW4 will be issued; that is, if any of the ICW4 operations are to be used.

ICW2

ICW2 is provided for compatibility with the 82C59A only. Its contents do not affect the operation of the interrupt bank in any way. Whenever the ICW2 of any of the three banks is written into, an interrupt is generated from bank A at level 1.5. The interrupt request will be cleared after the ICW2 register has been read by the 80376. The user is expected to program the corresponding vector register or to use it as an indicator that an attempt was made to alter the contents. Note that each ICW2 register has different addresses for read and write operations.

ICW3

The interrupt bank will only accept an ICW3 if programmed in the external cascade mode (as indicated in ICW1). ICW3 is used for specific programming within the cascade mode. The bits in ICW3 indicate which interrupt request inputs have a Slave cascaded to them. This will subsequently affect the interrupt vector generation during the interrupt acknowledge cycles as described previously.

ICW4

The ICW4 is accepted only if it was selected in ICW1. This command word register serves two functions:

- To select either the Automatic EOI mode or software EOI mode;
- To select if the Special Nested mode is to be used in conjunction with the cascade mode.

4.5.2 OPERATION CONTROL WORDS (OCW)

Once initialized by the ICW's, the interrupt banks will be operating in the Fully Nested Mode by default and they are ready to accept interrupt requests. However, the operations of each interrupt bank can be further controlled or modified by the use of OCW's. Three OCW's are available for programming various modes and commands. Note that all OCW's are 8-bit write-only registers.

The modes and operations controlled by the OCW's are:

- Fully Nested Mode;
- Rotating Priority Mode;
- Special Mask Mode;
- Poll Mode;
- EOI Commands;
- Read Status Commands.

OCW1

OCW1 is used solely for masking operations. It provides a direct link to the Internal Mask Register (IMR). The 80376 can write to this OCW register to enable or disable the interrupt inputs. Reading the pre-programmed mask can be done via the Interrupt Mask Register which will be discussed shortly.

OCW2

OCW2 is used to select End-Of-Interrupt, Automatic Priority Rotation, and Specific Priority Rotation operations. Associated commands and modes of these operations are selected using the different combinations of bits in OCW2.

Specifically, the OCW2 is used to:

- Designate an interrupt level (0–7) to be used to reset a specific ISR bit or to set a specific priority. This function can be enabled or disabled;
- Select which software EOI command (if any) is to be executed (i.e. Non-Specific or Specific EOI);
- Enable one of the priority rotation operations (i.e. Rotate On Non-Specific EOI, Rotate On Automatic EOI, or Rotate On Specific EOI).

OCW3

There are three main categories of operation that OCW3 controls. They are summarized as follows:

- To select and execute the Read Status Register Commands, either reading the Interrupt Request Register (IRR) or the In-Service Register (ISR);
- To issue the Poll Command. The Poll Command will override a Read Register Command if both functions are enabled simultaneously;
- To set or reset the Special Mask Mode.

4.5.3 POLL/INTERRUPT REQUEST/IN-SERVICE STATUS REGISTER

As the name implies, this 8-bit read-only register has multiple functions. Depending on the command issued in the OCW3, the content of this register reflects the result of the command executed. For a Poll Command, the register read contains the binary code of the highest priority level requesting service (if any). For a Read IRR Command, the register content will show the current pending interrupt request(s). Finally, for a Read ISR Command, this register will specify all interrupt levels which are being serviced.

4.5.4 INTERRUPT MASK REGISTER (IMR)

This is a read-only 8-bit register which, when read, will specify all interrupt levels within the same bank that are masked.

4.5.5 VECTOR REGISTERS (VR)

Each interrupt request input has an 8-bit read/write programmable vector register associated with it. The registers should be programmed to contain the interrupt vector for the corresponding request. The contents of the Vector Register will be placed on the Data Bus during the INTA cycles as described previously.

4.6 Programming

Programming the 82370 PIC is accomplished by using two types of command words: ICW's and OCW's. All modes and commands explained in the previous sections are programmable using the ICW's and OCW's. The ICW's are issued from the 80376 in a sequential format and are used to setup the banks in the 82370 PIC in an initial state of operation. The OCW's are issued as needed to vary and control the 82370 PIC's operations.

Both ICW's and OCW's are sent by the 80376 to the interrupt banks via the Data Bus. Each bank distinguishes between the different ICW's and OCW's by the I/O address map, the sequence they are issued (ICW's only), and by some dedicated bits among the ICW's and OCW's.

An example of programming the 82370 interrupt controllers is given in Appendix C (Programming the 82370 Interrupt Controllers).

All three interrupt banks are programmed in a similar way. Therefore, only a single bank will be described in the following sections.

4.6.1 INITIALIZATION (ICW)

Before normal operation can begin, each bank must be initialized by programming a sequence of two to four bytes written into the ICW's.

Figure 4-6 shows the initialization flow for an interrupt bank. Both ICW1 and ICW2 must be issued for any form of operation. However, ICW3 and ICW4 are used only if designated in ICW1. Once initialized, if any programming changes within the ICW's are to be made, the entire ICW sequence must be reprogrammed, not just an individual ICW.

Note that although the ICW2's in the 82370 PIC do not effect the Bank's operation, they still must be programmed in order to preserve the compatibility with the 82C59A. The contents programmed are not relevant to the overall operations of the interrupt banks. Also, whenever one of the three ICW2's is programmed, an interrupt level 1.5 in Bank A will be generated. This interrupt request will be cleared upon reading of the ICW2 registers. Since the three ICW2's share the same interrupt level and the system may not know the origin of the interrupt, all three ICW2's must be read.

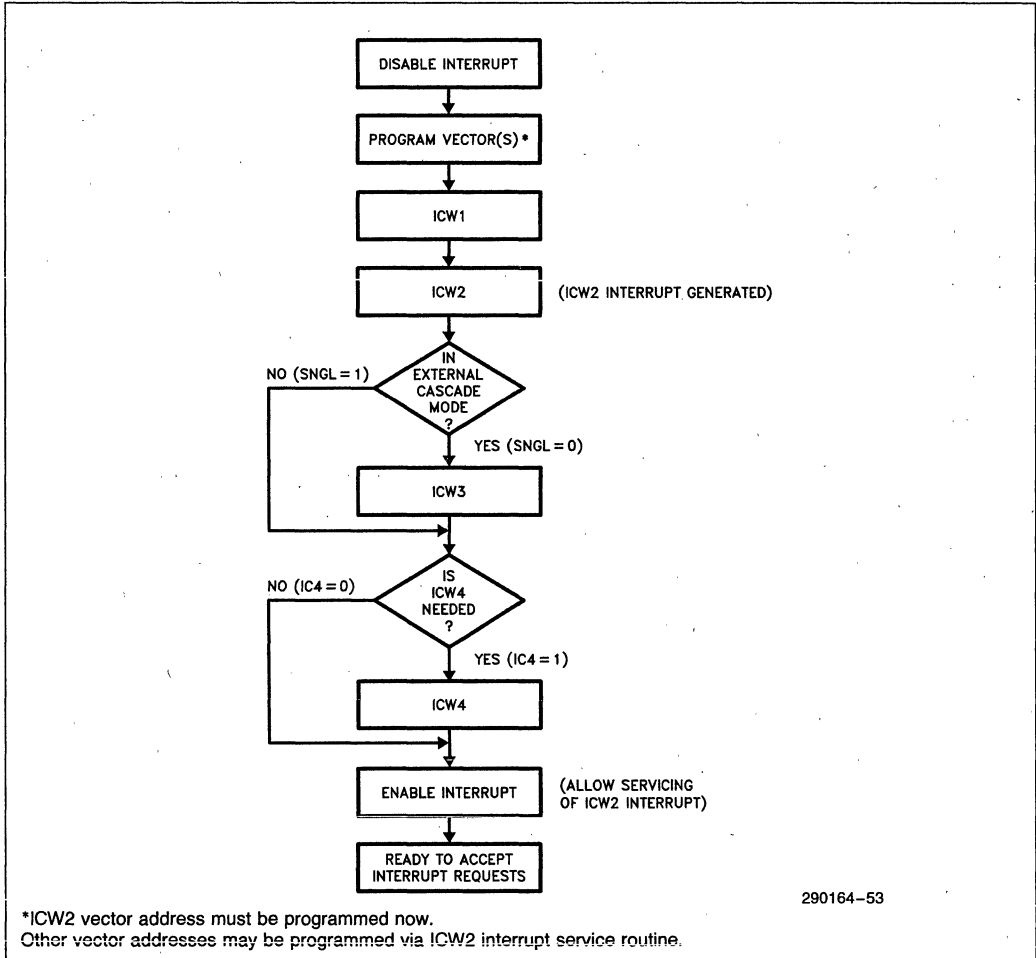


Figure 4-6. Initialization Sequence

Certain internal setup conditions occur automatically within the interrupt bank after the first ICW (ICW1) has been issued. These are:

- The edge sensitive circuit is reset, which means that following initialization, an interrupt request input must make a HIGH-to-LOW transition to generate an interrupt;
- The Interrupt Mask Register (IMR) is cleared; that is, all interrupt inputs are enabled;
- IRQ7 input of each bank is assigned priority 7 (lowest);
- Special Mask Mode is cleared and Status Read is set to IRR;
- If no ICW4 is needed, then no Automatic-EOI is selected.

4.6.2 VECTOR REGISTERS (VR)

Each interrupt request input has a separate Vector Register. These Vector Registers are used to store the pre-programmed vector number corresponding to their interrupt sources. In order to guarantee proper interrupt handling, all Vector Registers must be programmed with the predefined vector numbers. Since an interrupt request will be generated whenever an ICW2 is written during the initialization sequence, it is important that the Vector Register of IRQ1.5 in Bank A should be initialized and the interrupt service routine of this vector is set up before the ICW's are written.

4.6.3 OPERATION CONTROL WORDS (OCW)

After the ICW's are programmed, the operations of each interrupt controller bank can be changed by writing into the OCW's as explained before. There is no special programming sequence required for the OCW's. Any OCW may be written at any time in order to change the mode of or to perform certain operations on the interrupt banks.

Note that for reading IRR and ISR, there is no need to issue a Read Status Command to the OCW3 every time the IRR or ISR is to be read. Once a Read Status Command is received by the interrupt bank, it "remembers" which register is selected. However, this is not true when the Poll Command is used.

4.6.3.1 Read Status and Poll Commands (OCW3)

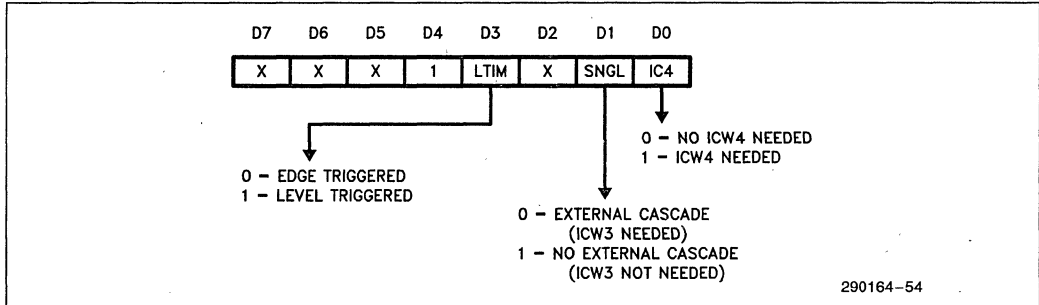
Since the reading of IRR and ISR status as well as the result of a Poll Command are available on the same read-only Status Register, a special Read Status/Poll Command must be issued before the Poll/Interrupt Request/In-Service Status Register is read. This command can be specified by writing the required control word into OCW3. As mentioned earlier, if both the Poll Command and the Status Read Command are enabled simultaneously, the Poll Command will override the Status Read. That is, after the command execution, the Status Register will contain the result of the Poll Command.

In the Poll Command, after the OCW3 is written, the 82370 PIC treats the next read to the Status Register as an interrupt acknowledge. This will set the appropriate IS bit if there is a request and read the priority level. Interrupt Request input status remains unchanged from the Poll Command to the Status Read.

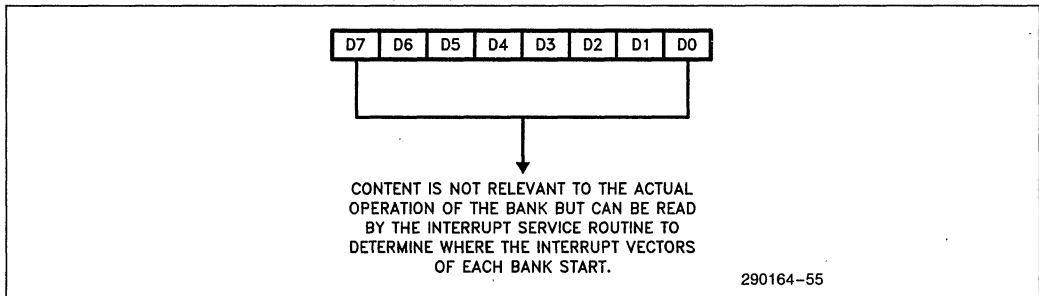
In addition to the above read commands, the Interrupt Mask Register (IMR) can also be read. When read, this register reflects the contents of the pre-programmed OCW1 which contains information on which interrupt request(s) is(are) currently disabled.

4.7 Register Bit Definition

INITIALIZATION COMMAND WORD 1 (ICW1)

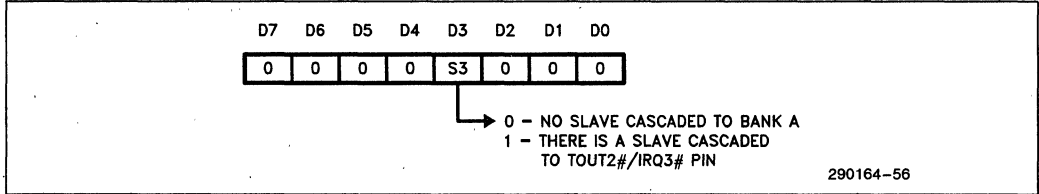


INITIALIZATION COMMAND WORD 2 (ICW2)

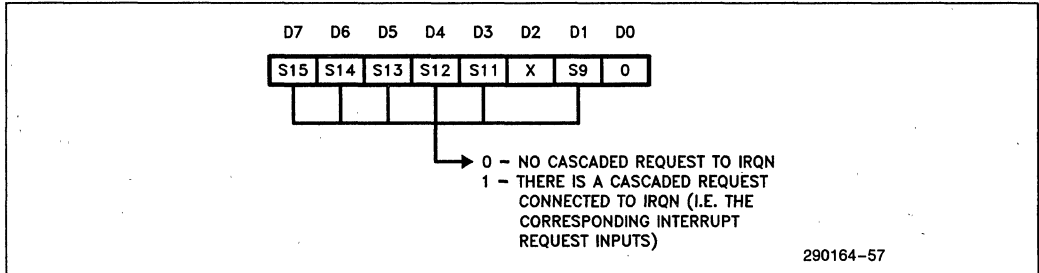


INITIALIZATION COMMAND WORD 3 (ICW3)

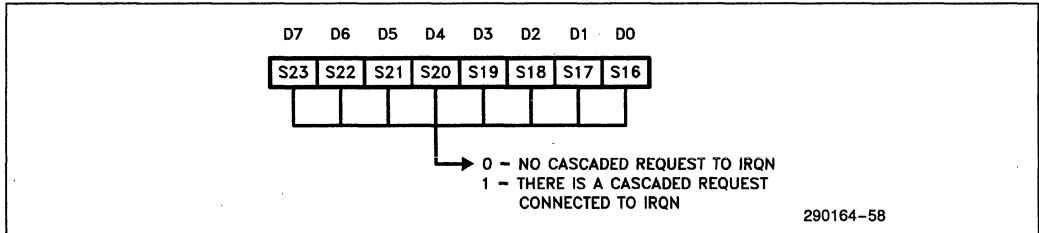
ICW3 for Bank A:



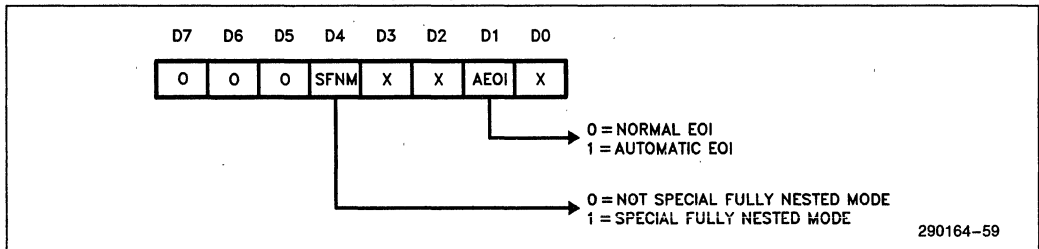
ICW3 for Bank B:



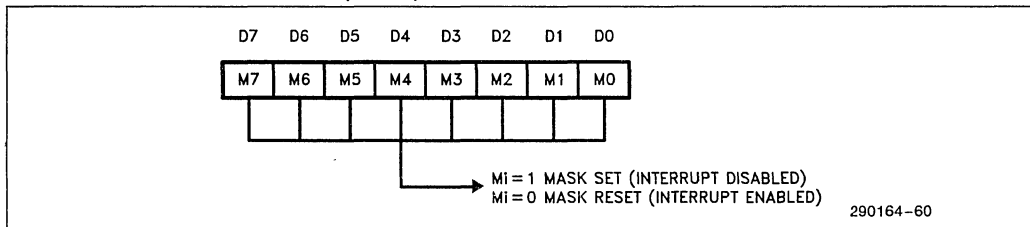
ICW3 for Bank C:



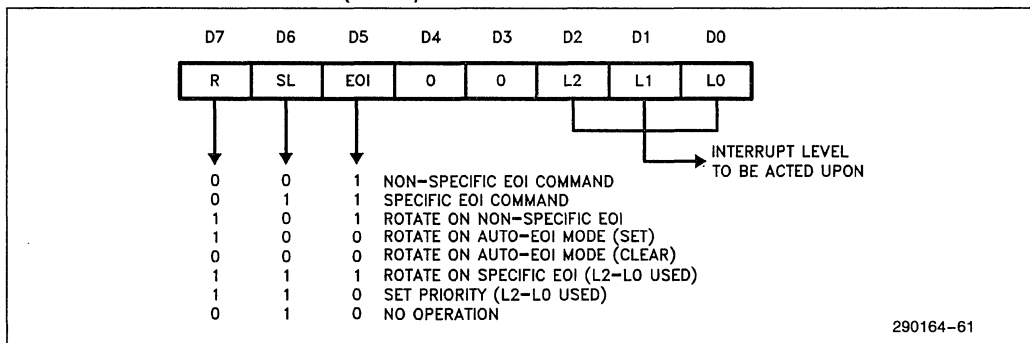
INITIALIZATION COMMAND WORD 4 (ICW4)



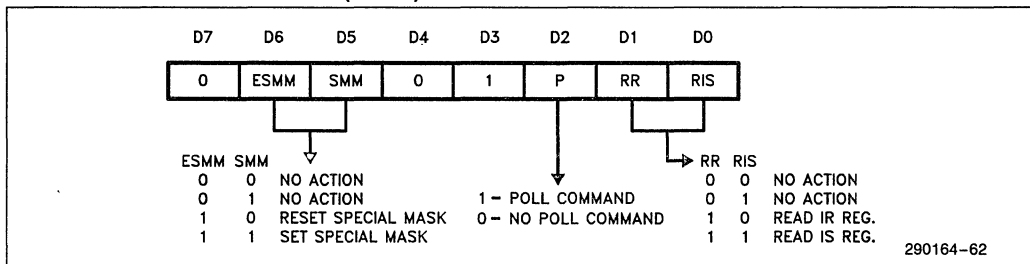
OPERATION CONTROL WORD 1 (OCW1)



OPERATION CONTROL WORD 2 (OCW2)



OPERATION CONTROL WORD 3 (OCW3)

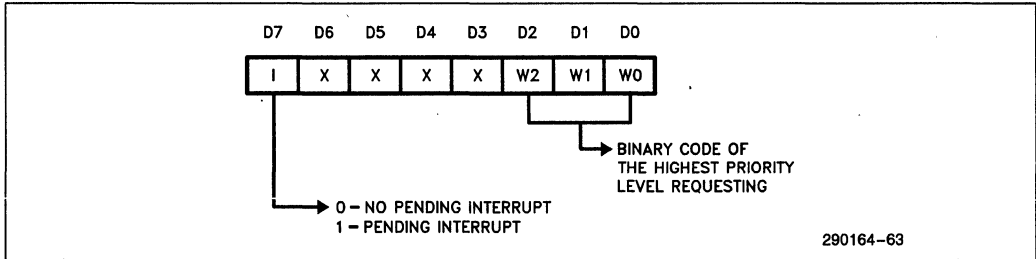


ESMM — Enable Special Mask Mode. When this bit is set to 1, it enables the SMM bit to set or reset the Special Mask Mode. When this bit is set to 0, SMM bit becomes don't care.

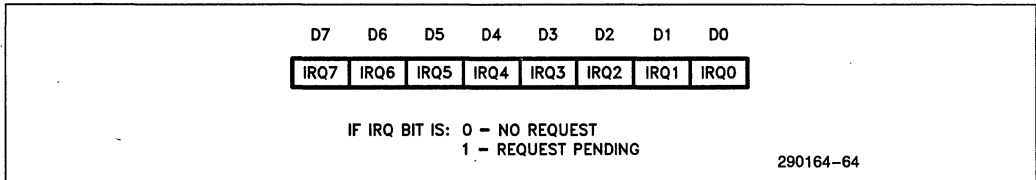
SMM — Special Mask Mode. If ESMM = 1 and SMM = 1, the interrupt controller bank will enter Special Mask Mode. If ESMM = 1 and SMM = 0, the bank will revert to normal mask mode. When ESMM = 0, SMM has no effect.

POLL/INTERRUPT REQUEST/IN-SERVICE STATUS REGISTER

Poll Command Status



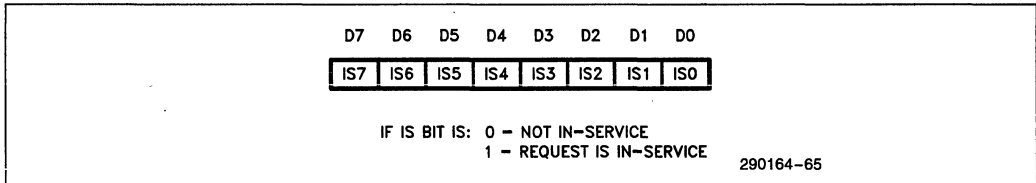
Interrupt Request Status



NOTE:

Although all Interrupt Request inputs are active LOW, the internal logical will invert the state of the pins so that when there is a pending interrupt request at the input, the corresponding IRQ bit will be set to HIGH in the Interrupt Request Status register.

In-Service Status



VECTOR REGISTER (VR)

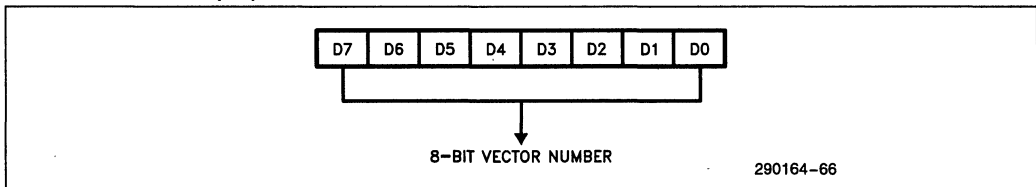


Table 4-4. Register Operational Summary

Operational Description	Command Words	Bits
Fully Nested Mode	OCW-Default	
Non-specific EOI Command	OCW2	EOI
Specific EOI Command	OCW2	SL, EOI, L0-L2
Automatic EOI Mode	ICW1, ICW4	IC4, AEOI
Rotate On Non-Specific EOI Command	OCW2	EOI
Rotate On Automatic EOI Mode	OCW2	R, SL, EOI
Set Priority Command	OCW2	L0-L2
Rotate On Specific EOI Command	OCW2	R, SL, EOI
Interrupt Mask Register	OCW1	M0-M7
Special Mask Mode	OCW3	ESMM, SMM
Level Triggered Mode	ICW1	LTIM
Edge Triggered Mode	ICW1	LTIM
Read Register Command, IRR	OCW3	RR, RIS
Read Register Command, ISR	OCW3	RR, RIS
Read IMR	IMR	M0-M7
Poll Command	OCW3	P
Special Fully Nested Mode	ICW1, ICW4	IC4, SFNM

4.8 Register Operational Summary

For ease of reference, Table 4-4 gives a summary of the different operating modes and commands with their corresponding registers.

5.0 PROGRAMMABLE INTERVAL TIMER

5.1 Functional Description

The 82370 contains four independently Programmable Interval Timers: Timer 0-3. All four timers are functionally compatible to the Intel 82C54. The first three timers (Timer 0-2) have specific functions. The fourth timer, Timer 3, is a general purpose timer. Table 5-1 depicts the functions of each timer. A brief description of each timer's function follows.

Table 5-1. Programmable Interval Timer Functions

Timer	Output	Function
0	IRQ8	Event Based IRQ8 Generator
1	TOUT1/REF #	Gen. Purpose/DRAM Refresh Req.
2	TOUT2/IRQ3 #	Gen. Purpose/Speaker Out/IRQ3 #
3	TOUT3 #	Gen. Purpose/IRQ0 Generator

TIMER 0—Event Based Interrupt Request 8 Generator

Timer 0 is intended to be used as an Event Counter. The output of this timer will generate an Interrupt Request 8 (IRQ8) upon a rising edge of the timer output (TOUT0). Normally, this timer is used to implement a time-of-day clock or system tick. The Timer 0 output is not available as an external signal.

TIMER 1—General Purpose/DRAM Refresh Request

The output of Timer 1, TOUT1, can be used as a general purpose timer or as a DRAM Refresh Request signal. The rising edge of this output creates a DRAM refresh request to the 82370 DRAM Refresh Controller. Upon reset, the Refresh Request function is disabled, and the output pin is the Timer 1 output.

TIMER 2—General Purpose/Speaker Out/IRQ3

The Timer 2 output, TOUT2#, could be used to support tone generation to an external speaker. This pin is a bidirectional signal. When used as an input, a logic LOW asserted at this pin will generate an Interrupt Request 3 (IRQ3 #) (see Programmable Interrupt Controller).

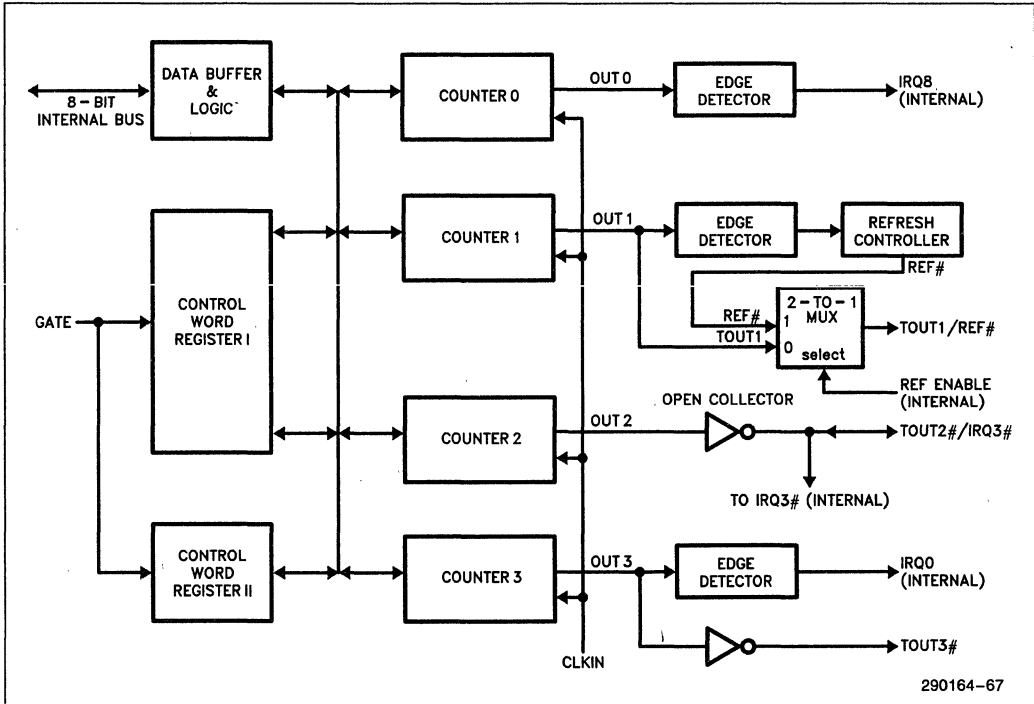


Figure 5-1. Block Diagram of Programmable Interval Timer

TIMER 3—General Purpose/Interrupt Request 0 Generator

The output of Timer 3 is fed to an edge detector and generates an Interrupt Request 0 (IRQ0) in the 82370. The inverted output of this timer (TOUT3#) is also available as an external signal for general purpose use.

5.1.1 INTERNAL ARCHITECTURE

The functional block diagram of the Programmable Interval Timer section is shown in Figure 5-1. Following is a description of each block.

DATA BUFFER & READ/WRITE LOGIC

This part of the Programmable Interval Timer is used to interface the four timers to the 82370 internal bus. The Data Buffer is for transferring commands and data between the 8-bit internal bus and the timers.

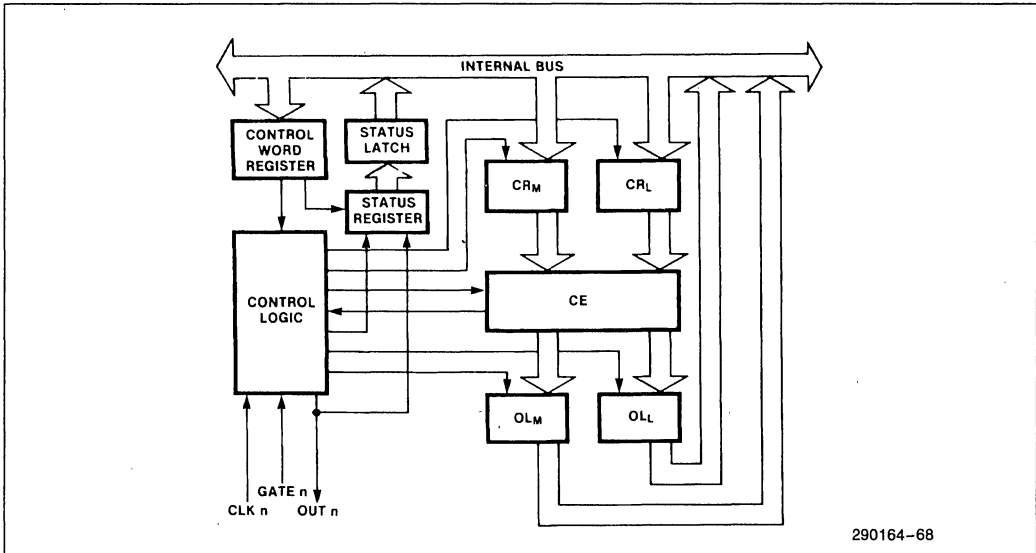
The Read/Write Logic accepts inputs from the internal bus and generates signals to control other functional blocks within the timer section.

CONTROL WORD REGISTERS I & II

The Control Word Registers are write-only registers. They are used to control the operating modes of the timers. Control Word Register I controls Timers 0, 1 and 2, and Control Word Register II controls Timer 3. Detailed description of the Control Word Registers will be included in the Register Set Overview section.

COUNTER 0, COUNTER 1, COUNTER 2, COUNTER 3

Counters 0, 1, 2, and 3 are the major parts of Timers 0, 1, 2, and 3, respectively. These four functional blocks are identical in operation, so only a single counter will be described. The internal block diagram of one counter is shown in Figure 5-2.



290164-68

Figure 5-2. Internal Block Diagram of a Counter

The four counters share a common clock input (CLKIN), but otherwise are fully independent. Each counter is programmable to operate in a different mode.

Although the Control Word Register is shown in the figure, it is not part of the counter itself. Its programmed contents are used to control the operations of the counters.

The Status Register, when latched, contains the current contents of the Control Word Register and status of the output and Null Count Flag (see Read Back Command).

The Counting Element (CE) is the actual counter. It is a 16-bit presettable synchronous down counter.

The Output Latches (OL) contain two 8-bit latches (OLM and OLL). Normally, these latches "follow" the content of the CE. OLM contains the most significant byte of the counter and OLL contains the least significant byte. If the Counter Latch Command is sent to the counter, OL will latch the present count until read by the 80376 and then return to follow the CE. One latch at a time is enabled by the timer's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that CE cannot be read. Whenever the count is read, it is one of the OL's that is being read.

When a new count is written into the counter, the value will be stored in the Count Registers (CR), and transferred to CE. The transferring of the contents from CR's to CE is defined as "loading" of the counter. The Count Register contains two 8-bit registers: CRM (which contains the most significant byte) and CRL (which contains the least significant byte). Similar to the OL's, the Control Logic allows one register at a time to be loaded from the 8-bit internal bus. However, both bytes are transferred from the CR's to the CE simultaneously. Both CR's are cleared when the Counter is programmed. This way, if the Counter has been programmed for one byte count (either the most significant or the least significant byte only), the other byte will be zero. Note that CE cannot be written into directly. Whenever a count is written, it is the CR that is being written.

As shown in the diagram, the Control Logic consists of three signals: CLKIN, GATE, and OUT. CLKIN and GATE will be discussed in detail in the section that follows. OUT is the internal output of the counter. The external outputs of some timers (TOUT) are the inverted version of OUT (see TOUT1, TOUT2#, TOUT3#). The state of OUT depends on the mode of operation of the timer.

5.2 Interface Signals

5.2.1 CLKIN

CLKIN is an input signal used by all four timers for internal timing reference. This signal can be independent of the 82370 system clock, CLK2. In the following discussion, each "CLK Pulse" is defined as the time period between a rising edge and a falling edge, in that order, of CLKIN.

During the rising edge of CLKIN, the state of GATE is sampled. All new counts are loaded and counters are decremented on the falling edge of CLKIN.

5.2.2 TOUT1, TOUT2#, TOUT3#

TOUT1, TOUT2# and TOUT3# are the external output signals of Timer 1, Timer 2 and Timer 3, respectively. TOUT2# and TOUT3# are the inverted signals of their respective counter outputs, OUT. There is no external output for Timer 0.

If Timer 2 is to be used as a tone generator of a speaker, external buffering must be used to provide sufficient drive capability.

The Outputs of Timer 2 and 3 are dual function pins. The output pin of Timer 2 (TOUT2#/IRQ3#), which is a bidirectional open-collector signal, can also be used as interrupt request input. When the interrupt function is enabled (through the Programmable Interrupt Controller), a LOW on this input will generate an Interrupt Request 3# to the 82370 Programmable Interrupt Controller. This pin has a weak internal pull-up resistor. To use the IRQ3# function, Timer 2 should be programmed so that OUT2 is LOW. Additionally, OUT3 of Timer 3 is connected to an edge detector which will generate an Interrupt Request 0 (IRQ0) to the 82370 after the rising edge of OUT3 (see Figure 5-1).

5.2.3 GATE

GATE is not an externally controllable signal. Rather, it can be software controlled with the Internal Control Port. The state of GATE is always sampled on the rising edge of CLKIN. Depending on the mode of operation, GATE is used to enable/disable counting or trigger the start of an operation.

For Timer 0 and 1, GATE is always enabled (HIGH). For Timer 2 and 3, GATE is connected to Bit 0 and 6, respectively, of an Internal Control Port (at address 61H) of the 82370. After a hardware reset, the state of GATE of Timer 2 and 3 is disabled (LOW).

5.3 Modes of Operation

Each timer can be independently programmed to operate in one of six different modes. Timers are programmed by writing a Control Word into the Control Word Register followed by an Initial Count (see Programming).

The following are defined for use in describing the different modes of operation.

CLK Pulse— A rising edge, then a falling edge, in that order, of CLKIN.

Trigger— A rising edge of a timer's GATE input.

Timer/Counter Loading— The transfer of a count from Count Register (CR) to Count Element (CE).

5.3.1 MODE 0—INTERRUPT ON TERMINAL COUNT

Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially LOW, and will remain LOW until the counter reaches zero. OUT then goes HIGH and remains HIGH until a new count or a new Mode 0 Control Word is written into the counter.

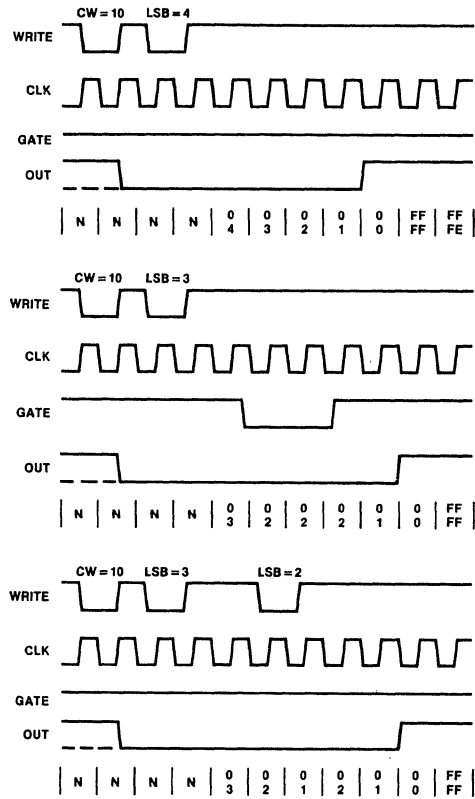
In this mode, GATE = HIGH enables counting; GATE = LOW disables counting. However, GATE has no effect on OUT.

After the Control Word and initial count are written to a timer, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go HIGH until N + 1 CLK pulses after the initial count is written.

If a new count is written to the timer, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1. Writing the first byte disables counting, OUT is set LOW immediately (i.e. no CLK pulse required).
2. Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the counting sequence to be synchronized by software. Again, OUT does not go HIGH until N + 1 CLK pulses after the new count of N is written.



290164-69

NOTES:

The following conventions apply to all mode timing diagrams.

1. Counters are programmed for binary (not BCD) counting and for reading/writing least significant byte (LSB) only.
2. The counter is always selected (CS# always low).
3. CW stands for "Control Word"; CW = 10 means a control word of 10, Hex is written to the counter.
4. LSB stands for "Least significant byte" of count.
5. Numbers below diagrams are count values.

The lower number is the least significant byte.

The upper number is the most significant byte. Since the counter is programmed to read/write LSB only, the most significant byte cannot be read.

N stands for an undefined count.

Vertical lines show transitions between count values.

Figure 5-3. Mode 0

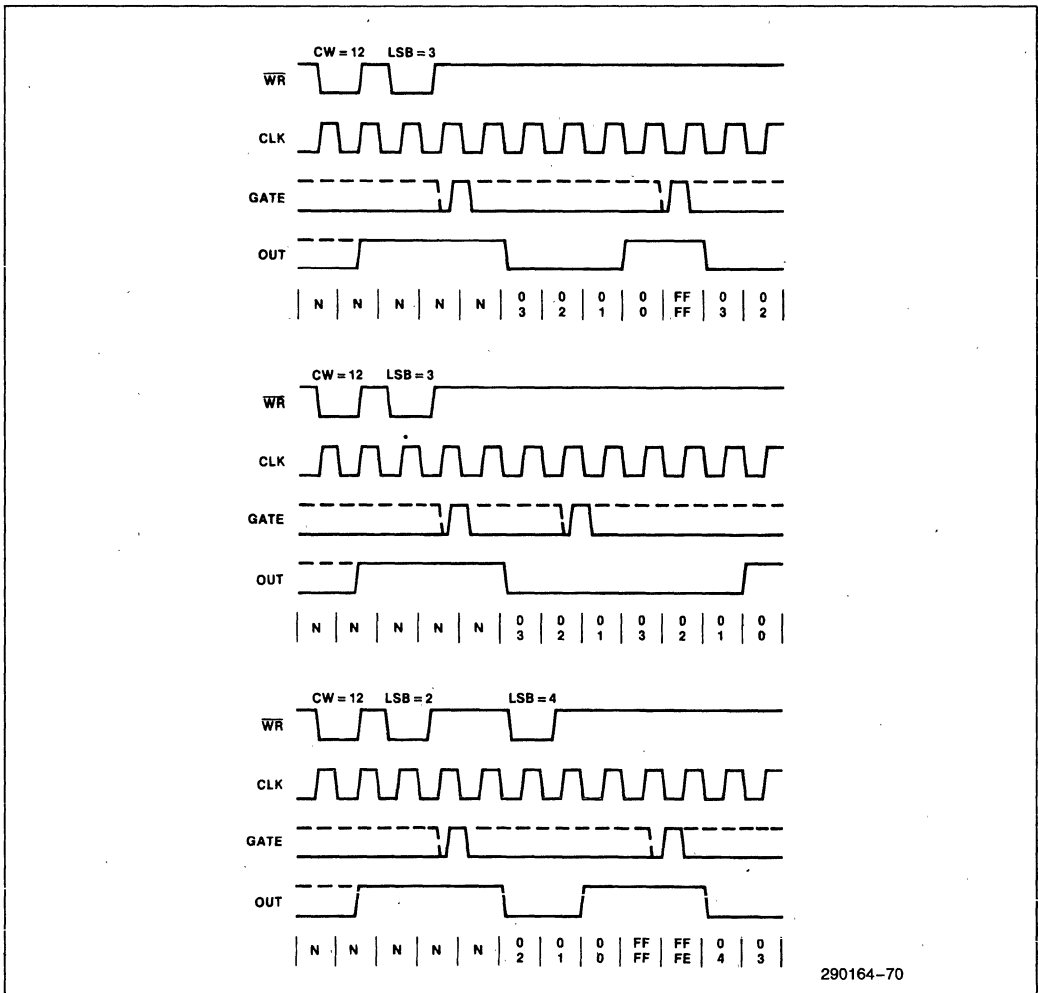
If an initial count is written while GATE is LOW, the counter will be loaded on the next CLK pulse. When GATE goes HIGH, OUT will go HIGH N CLK pulses later; no CLK pulse is needed to load the counter as this has already been done.

5.3.2 MODE 1—GATE RETRIGGERABLE ONE-SHOT

In this mode, OUT will be initially HIGH. OUT will go LOW on the CLK pulse following a trigger to start the

one-shot operation. The OUT signal will then remain LOW until the timer reaches zero. At this point, OUT will stay HIGH until the next trigger comes in. Since the state of GATE signals of Timer 0 and 1 are internally set to HIGH.

After writing the Control Word and initial count, the timer is considered "armed". A trigger results in loading the timer and setting OUT LOW on the next CLK pulse. Therefore, an initial count of N will result in a one-shot pulse width of N CLK cycles. Note



290164-70

Figure 5-4. Mode 1

that this one-shot operation is retriggerable; i.e. OUT will remain LOW for N CLK pulses after every trigger. The one-shot operation can be repeated without re-writing the same count into the timer.

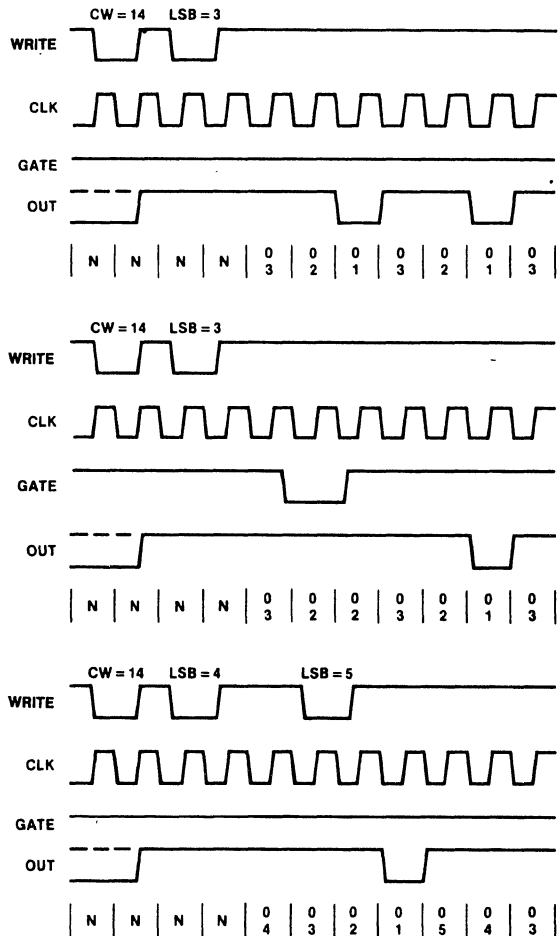
If a new count is written to the timer during a one-shot operation, the current one-shot pulse width will not be affected until the timer is retriggered. This is because loading of the new count to CE will occur only when the one-shot is triggered.

5.3.3 MODE 2-RATE GENERATOR

This mode is a divide-by-N counter. It is typically used to generate a Real Time Clock interrupt. OUT will initially be HIGH. When the initial count has dec-

remented to 1, OUT goes LOW for one CLK pulse, then OUT goes HIGH again. Then the timer reloads the initial count and the process is repeated. In other words, this mode is periodic since the same sequence is repeated itself indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.

Similar to Mode 0, GATE=HIGH enables counting, where GATE=LOW disables counting. If GATE goes LOW during an output pulse (LOW), OUT is set HIGH immediately. A trigger (rising edge on GATE) will reload the timer with the initial count on the next CLK pulse. Then, OUT will go LOW (for one CLK pulse) N CLK pulses after the new trigger. Thus, GATE can be used to synchronize the timer.



290164-71

NOTE:
A GATE transition should not occur one clock prior to terminal count.

Figure 5-5. Mode 2

After writing a Control Word and initial count, the timer will be loaded on the next CLK pulse. OUT goes LOW (for one CLK pulse) N CLK pulses after the initial count is written. This is another way the timer may be synchronized by software.

Writing a new count while counting does not affect the current counting sequence because the new count will not be loaded until the end of the current counting cycle. If a trigger is received after writing a

new count but before the end of the current period, the timer will be loaded with the new count on the next CLK pulse after the trigger, and counting will continue with the new count.

5.3.4 MODE 3—SQUARE WAVE GENERATOR

Mode 3 is typically used for Baud Rate generation. Functionally, this mode is similar to Mode 2 except

for the duty cycle of OUT. In this mode, OUT will be initially HIGH. When half of the initial count has expired, OUT goes low for the remainder of the count. The counting sequence will be repeated, thus this mode is also periodic. Note that an initial count of N results in a square wave with a period of N CLK pulses.

The GATE input can be used to synchronize the timer. GATE=HIGH enables counting; GATE=LOW disables counting. If GATE goes LOW while OUT is LOW, OUT is set HIGH immediately (i.e. no CLK pulse is required). A trigger reloads the timer with the initial count on the next CLK pulse.

After writing a Control Word and initial count, the timer will be loaded on the next CLK pulse. This allows the timer to be synchronized by software.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the timer will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

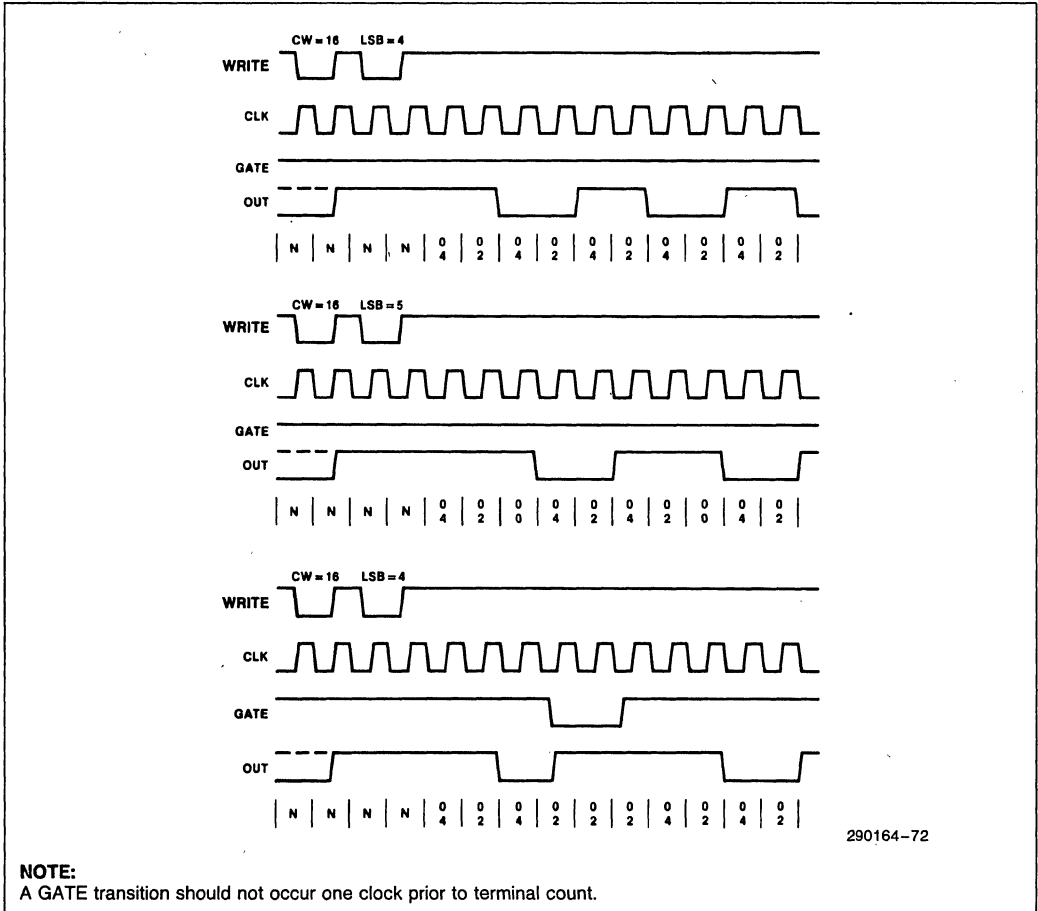
There is a slight difference in operation depending on whether the initial count is EVEN or ODD. The following description is to show exactly how this mode is implemented.

EVEN COUNTS:

OUT is initially HIGH. The initial count is loaded on one CLK pulse and is decremented by two on succeeding CLK pulses. When the count expires (decremented to 2), OUT changes to LOW and the timer is reloaded with the initial count. The above process is repeated indefinitely.

ODD COUNTS:

OUT is initially HIGH. The initial count minus one (which is an even number) is loaded on one CLK pulse and is decremented by two on succeeding CLK pulses. One CLK pulse after the count expires (decremented to 2), OUT goes LOW and the timer is loaded with the initial count minus one again. Succeeding CLK pulses decrement the count by two. When the count expires, OUT goes HIGH immediately and the timer is reloaded with the initial count minus one. The above process is repeated indefinitely. So for ODD counts, OUT will HIGH for $(N+1)/2$ counts and LOW for $(N-1)/2$ counts.



NOTE:
A GATE transition should not occur one clock prior to terminal count.

Figure 5-6. Mode 3

5.3.5 MODE 4—INITIAL COUNT TRIGGERED STROBE

This mode allows a strobe pulse to be generated by writing an initial count to the timer. Initially, OUT will be HIGH. When a new initial count is written into the timer, the counting sequence will begin. When the initial count expires (decremented to 1), OUT will go LOW for one CLK pulse and then go HIGH again.

Again, GATE=HIGH enables counting while GATE = LOW disables counting. GATE has no effect on OUT.

After writing the Control Word and initial count, the timer will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe LOW until N+1 CLK pulses after initial count is written.

If a new count is written during counting, it will be loaded in the next CLK pulse and counting will continue from the new count.

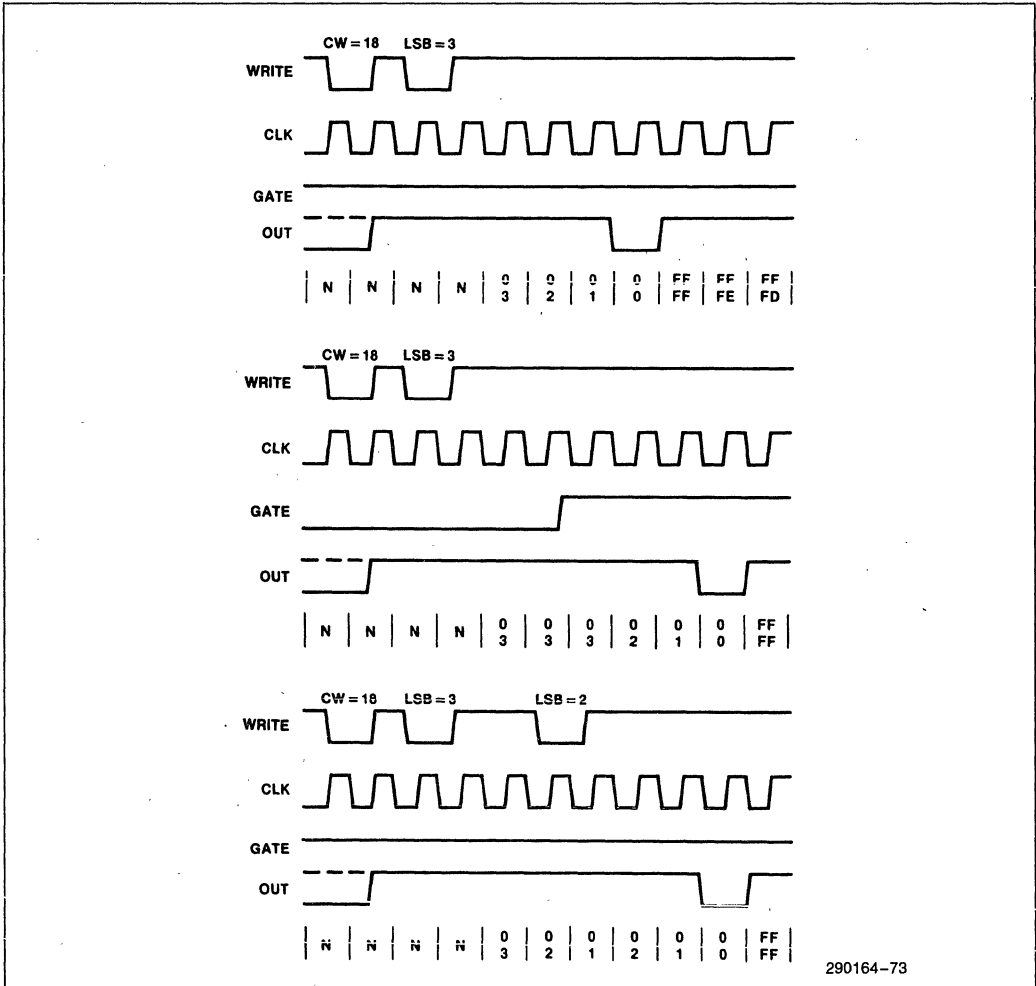


Figure 5-7. Mode 4

If a two-byte count is written, the following will occur:

1. Writing the first byte has no effect on counting.
2. Writing the second byte allows the new count to be loaded on the next CLK pulse.

OUT will strobe LOW N+1 CLK pulses after the new count of N is written. Therefore, when the strobe pulse will occur after a trigger depends on the value of the initial count loaded.

by writing an initial count. Initially, OUT will be HIGH. Counting is triggered by a rising edge of GATE. When the initial count has expired (decremented to 1), OUT will go LOW for one CLK pulse and then go HIGH again.

After loading the Control Word and initial count, the Count Element will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count. Therefore, for an initial count of N, OUT does not strobe LOW until N+1 CLK pulses after a trigger.

5.3.6 MODE 5-GATE RETRIGGERABLE STROBE

Mode 5 is very similar to Mode 4 except the count sequence is triggered by the gate signal instead of

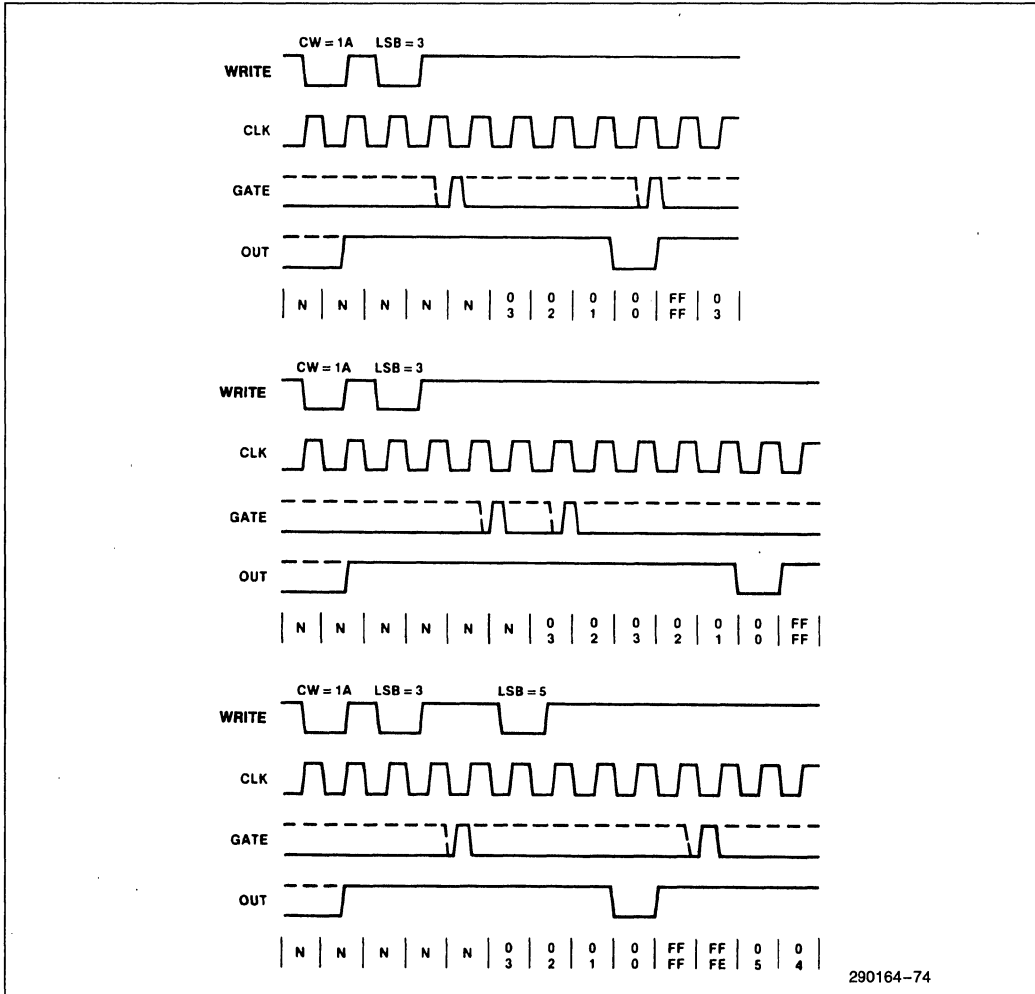


Figure 5-8. Mode 5

The counting sequence is retriggerable. Every trigger will result in the timer being loaded with the initial count on the next CLK pulse.

If the new count is written during counting, the current counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the timer will be loaded with the new count on the next CLK pulse and a new count sequence will start from there.

5.3.7 OPERATION COMMON TO ALL MODES

5.3.7.1 GATE

The GATE input is always sampled on the rising edge of CLKIN. In Modes 0, 2, 3 and 4, the GATE input is level sensitive. The logic level is sampled on the rising edge of CLKIN. In Modes 1, 2, 3 and 5, the GATE input is rising edge sensitive. In these modes,

Summary of Gate Operations

Mode	GATE LOW or Going LOW	GATE Rising	HIGH
0	Disable count	No Effect	Enable count
1	No Effect	1. Initiate count 2. Reset output after next clock	No Effect
2	1. Disable count 2. Sets output HIGH immediately	Initiate count	Enable count
3	1. Disable count 2. Sets output HIGH immediately	initiate count	Enable count
4	Disable count	No Effect	Enable count
5	No Effect	Initiate count	No Effect

a rising edge of GATE (trigger) sets an edge sensitive flip-flop in the timer. The flip-flop is reset immediately after it is sampled. This way, a trigger will be detected no matter when it occurs; i.e. a HIGH logic level does not have to be maintained until the next rising edge of CLKIN. Note that in Modes 2 and 3, the GATE input is both edge and level sensitive.

5.3.7.2 Counter

New counts are loaded and counters are decremented on the falling edge of CLKIN. The largest possible initial count is 0. This is equivalent to 2**16 for binary counting and 10**4 for BCD counting.

Note that the counter does not stop when it reaches zero. In Modes 0, 1, 4 and 5, the counter 'wraps around' to the highest count: either FFFF Hex for binary counting or 9999 for BCD counting, and continues counting. Modes 2 and 3 are periodic. The counter reloads itself with the initial count and continues counting from there.

The minimum and maximum initial count in each counter depends on the mode of operation. They are summarized below.

Mode	Min	Max
0	1	0
1	1	0
2	2	0
3	2	0
4	1	0
5	1	0

5.4 Register Set Overview

The Programmable Interval Timer module of the 82370 contains a set of six registers. The port address map of these registers is shown in Table 5-2.

Table 5-2. Timer Register Port Address Map

Port Address	Description
40H	Counter 0 Register (read/write)
41H	Counter 1 Register (read/write)
42H	Counter 2 Register (read/write)
43H	Control Word Register I (Counter 0, 1 & 2) (write-only)
44H	Counter 3 Register (read/write)
45H	Reserved
46H	Reserved
47H	Control Word Register II (Counter 3) (write-only)

5.4.1 COUNTER 0, 1, 2, 3 REGISTER

These four 8-bit registers are functionally identical. They are used to write the initial count value into the respective timer. Also, they can be used to read the latched count value of a timer. Since they are 8-bit registers, reading and writing of the 16-bit initial count must follow the count format specified in the Control Word Registers; i.e. least significant byte only, most significant byte only, or least significant byte then most significant byte (see Programming).

5.4.2 CONTROL WORD REGISTER I & II

There are two Control Word Registers associated with the Timer section. One of the two registers (Control Word Register I) is used to control the operations of Counters 0, 1 and 2 and the other (Control Word Register II) is for Counter 3. The major functions of both Control Word Registers are listed below:

- Select the timer to be programmed.
- Define which mode the selected timer is to operate in.
- Define the count sequence; i.e. if the selected timer is to count as a Binary Counter or a Binary Coded Decimal (BCD) Counter.
- Select the byte access sequence during timer read/write operations; i.e. least significant byte only, most significant only, or least significant byte first, then most significant byte.

Also, the Control Word Registers can be programmed to perform a Counter Latch Command or a Read Back Command which will be described later.

5.5 Programming

5.5.1 INITIALIZATION

Upon power-up or reset, the state of all timers is undefined. The mode, count value, and output of all timers are random. From this point on, how each timer operates is determined solely by how it is programmed. Each timer must be programmed before it can be used. Since the outputs of some timers can generate interrupt signals to the 82370, all timers should be initialized to a known state.

Counters are programmed by writing a Control Word into their respective Control Word Registers. Then, an Initial Count can be written into the corresponding Count Register. In general, the programming procedure is very flexible. Only two conventions need to be remembered:

1. For each timer, the Control Word must be written before the initial count is written.
2. The 16-bit initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte first, followed by most significant byte).

Since the two Control Word Registers and the four Counter Registers have separate addresses, and each timer can be individually selected by the appropriate Control Word Register, no special instruction sequence is required. Any programming sequence that follows the conventions above is acceptable.

A new initial count may be written to a timer at any time without affecting the timer's programmed mode in any way. Count sequence will be affected as described in the Modes of Operation section. Note that the new count must follow the programmed count format.

If a timer is previously programmed to read/write two-byte counts, the following precaution applies. A program must not transfer control between writing the first and second byte to another routine which also writes into the same timer. Otherwise, the read/write will result in incorrect count.

Whenever a Control Word is written to a timer, all control logic for that timer(s) is immediately reset (i.e. no CLK pulse is required). Also, the corresponding output in, TOUT#, goes to a known initial state.

5.5.2 READ OPERATION

Three methods are available to read the current count as well as the status of each timer. They are: Read Counter Registers, Counter Latch Command and Read Back Command. Below is a description of these methods.

READ COUNTER REGISTERS

The current count of a timer can be read by performing a read operation on the corresponding Counter Register. The only restriction of this read operation is that the CLKIN of the timers must be inhibited by using external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result. Note that since all four timers are sharing the same CLKIN signal, inhibiting CLKIN to read a timer will unavoidably disable the other timers also. This may prove to be impractical. Therefore, it is suggested that either the Counter Latch Command or the Read Back Command can be used to read the current count of a timer.

Another alternative is to temporarily disable a timer before reading its Counter Register by using the GATE input. Depending on the mode of operation, GATE=LOW will disable the counting operation. However, this option is available on Timer 2 and 3 only, since the GATE signals of the other two timers are internally enabled all the time.

COUNTER LATCH COMMAND

A Counter Latch Command will be executed whenever a special Control Word is written into a Control Word Register. Two bits written into the Control Word Register distinguish this command from a 'regular' Control Word (see Register Bit Definition). Also, two other bits in the Control Word will select which counter is to be latched.

Upon execution of this command, the selected counter's Output Latch (OL) latches the count at the time the Counter Latch Command is received. This

count is held in the latch until it is read by the 80376, or until the timer is reprogrammed. The count is then unlatched automatically and the OL returns to "following" the Counting Element (CE). This allows reading the contents of the counters "on the fly" without affecting counting in progress. Multiple Counter Latch Commands may be used to latch more than one counter. Each latched count is held until it is read. Counter Latch Commands do not affect the programmed mode of the timer in any way.

If a counter is latched, and at some time later, it is latched again before the prior latched count is read, the second Counter Latch Command is ignored. The count read will then be the count at the time the first command was issued.

In any event, the latched count must be read according to the programmed format. Specifically, if the timer is programmed for two-byte counts, two bytes must be read. However, the two bytes do not have to be read right after the other. Read/write or programming operations of other timers may be performed between them.

Another feature of this Counter Latch Command is that read and write operations of the same timer may be interleaved. For example, if the timer is programmed for two-byte counts, the following sequence is valid.

1. Read least significant byte.
2. Write new least significant byte.
3. Read most significant byte.
4. Write new most significant byte.

If a timer is programmed to read/write two-byte counts, the following precaution applies. A program must not transfer control between reading the first and second byte to another routine which also reads from that same timer. Otherwise, an incorrect count will be read.

READ BACK COMMAND

The Read Back Command is another special Command Word operation which allows the user to read the current count value and/or the status of the selected timer(s). Like the Counter Latch Command, two bits in the Command Word identify this as a Read Back Command (see Register Bit Definition).

The Read Back Command may be used to latch multiple counter Output Latches (OL's) by selecting more than one timer within a Command Word. This single command is functionally equivalent to several Counter Latch Commands, one for each counter to

be latched. Each counter's latched count will be held until it is read by the 80376 or until the timer is reprogrammed. The counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple Read Back commands are issued to the same timer without reading the count, all but the first are ignored; i.e. the count read will correspond to the very first Read Back Command issued.

As mentioned previously, the Read Back Command may also be used to latch status information of the selected timer(s). When this function is enabled, the status of a timer can be read from the Counter Register after the Read Back Command is issued. The status information of a timer includes the following:

1. Mode of timer:

This allows the user to check the mode of operation of the timer last programmed.

2. State of TOUT pin of the timer:

This allows the user to monitor the counter's output pin via software, possibly eliminating some hardware from a system.

3. Null Count/Count available:

The Null Count Bit in the status byte indicates if the last count written to the Count Register (CR) has been loaded into the Counting Element (CE). The exact time this happens depends on the mode of the timer and is described in the Programming section. Until the count is loaded into the Counting Element (CE), it cannot be read from the timer. If the count is latched or read before this occurs, the count value will not reflect the new count just written.

If multiple status latch operations of the timer(s) are performed without reading the status, all but the first command are ignored; i.e. the status read in will correspond to the first Read Back Command issued.

Both the current count and status of the selected timer(s) may be latched simultaneously by enabling both functions in a single Read Back Command. This is functionally the same as issuing two separate Read Back Commands at once. Once again, if multiple read commands are issued to latch both the count and status of a timer, all but the first command will be ignored.

If both count and status of a timer are latched, the first read operation of that timer will return the latched status, regardless of which was latched first. The next one or two (if two count bytes are to be read) read operations return the latched count. Note that subsequent read operations on the Counter Register will return the unlatched count (like the first read method discussed).

5.6 Register Bit Definitions

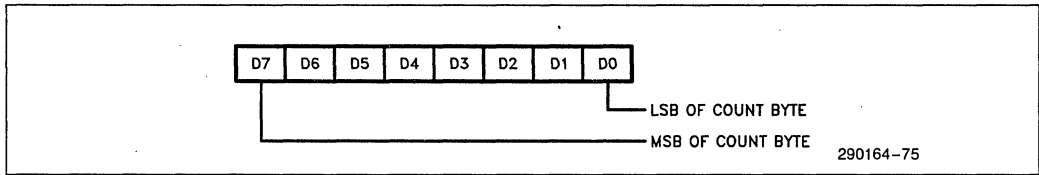
COUNTER 0, 1, 2, 3 REGISTER (READ/WRITE)

Port Address	Description
40H	Counter 0 Register (read/write)
41H	Counter 1 Register (read/write)
42H	Counter 2 Register (read/write)
44H	Counter 3 Register (read/write)
45H	Reserved
46H	Reserved

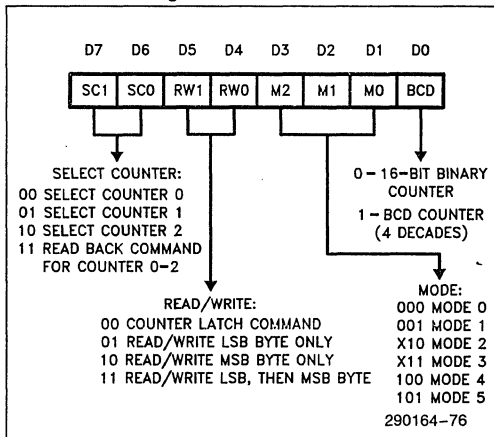
Note that these 8-bit registers are for writing and reading of one byte of the 16-bit count value, either the most significant or the least significant byte.

CONTROL WORD REGISTER I & II (WRITE-ONLY)

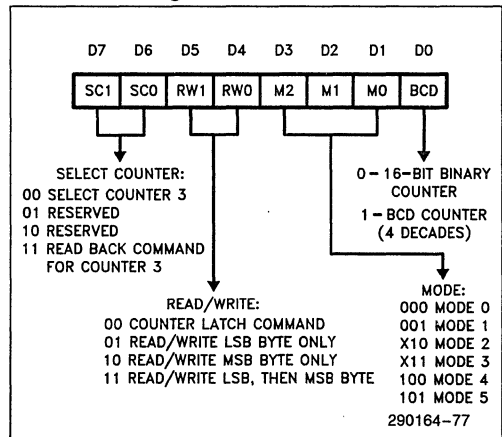
Port Address	Description
43H	Control Word Register I (Counter 0, 1, 2 (write-only))
47H	Control Word Register II (Counter 3) (write-only)



Control Word Register I

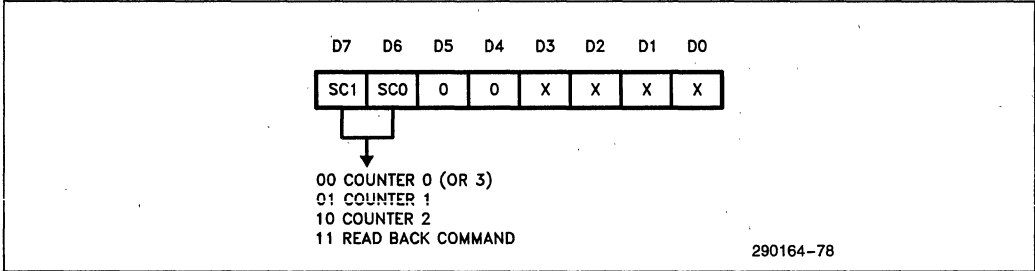


Control Word Register II



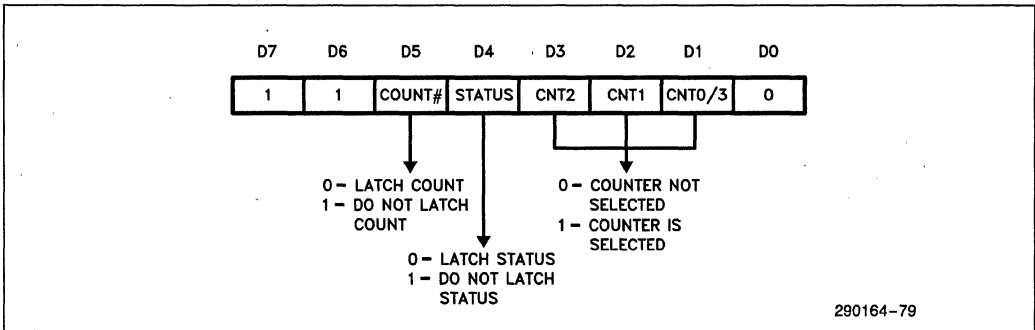
COUNTER LATCH COMMAND FORMAT

(Write to Control Word Register)



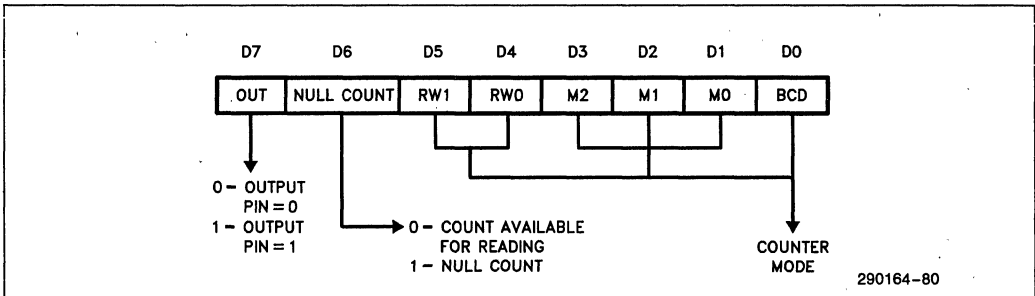
READ BACK COMMAND FORMAT

(Write to Control Word Register)



STATUS FORMAT

(Returned from Read Back Command)



6.0 WAIT STATE GENERATOR

6.1 Functional Description

The 82370 contains a programmable Wait State Generator which can generate a pre-programmed number of wait states during both CPU and DMA initiated bus cycles. This Wait State Generator is capable of generating 1 to 16 wait states in non-pipelined mode, and 0 to 15 wait states in pipelined mode. Depending on the bus cycle type and the two Wait State Control inputs (WSC 0-1), a pre-programmed number of wait states in the selected Wait State Register will be generated.

The Wait State Generator can also be disabled to allow the use of devices capable of generating their own READY# signals. Figure 6-1 is a block diagram of the Wait State Generator.

6.2 Interface Signals

The following describes the interface signals which affect the operation of the Wait State Generator. The READY#, WSC0 and WSC1 signals are inputs. READYO# is the ready output signal to the host processor.

6.2.1 READY#

READY# is an active LOW input signal which indicates to the 82370 the completion of a bus cycle. In the Master mode (e.g. 82370 initiated DMA transfer), this signal is monitored to determine whether a peripheral or memory needs wait states inserted in the current bus cycle. In the Slave mode, it is used (together with the ADS# signal) to trace CPU bus cycles to determine if the current cycle is pipelined.

6.2.2 READYO#

READYO# (Ready Out#) is an active LOW output signal and is the output of the Wait State Generator. The number of wait states generated depends on the WSC(0-1) inputs. Note that special cases are handled for access to the 82370 internal registers and for the Refresh cycles. For 82370 internal register access, READYO# will be delayed to take into the command recovery time of the register. One or more wait states will be generated in a pipelined cycle. During refresh, the number of wait states will be determined by the preprogrammed value in the Refresh Wait State Register.

In the simplest configuration, READYO# can be connected to the READY# input of the 82370 and the 80376 CPU. This is, however, not always the case. If external circuitry is to control the READY# inputs as well, additional logic will be required (see Application Issues).

6.2.3 WSC(0-1)

These two Wait State Control inputs, together with the M/IO# input, select one of the three pre-programmed 8-bit Wait State Registers which determines the number of wait states to be generated. The most significant half of the three Wait State Registers corresponds to memory accesses, the least significant half to I/O accesses. The combination WSC(0-1) = 11 disables the Wait State Generator.

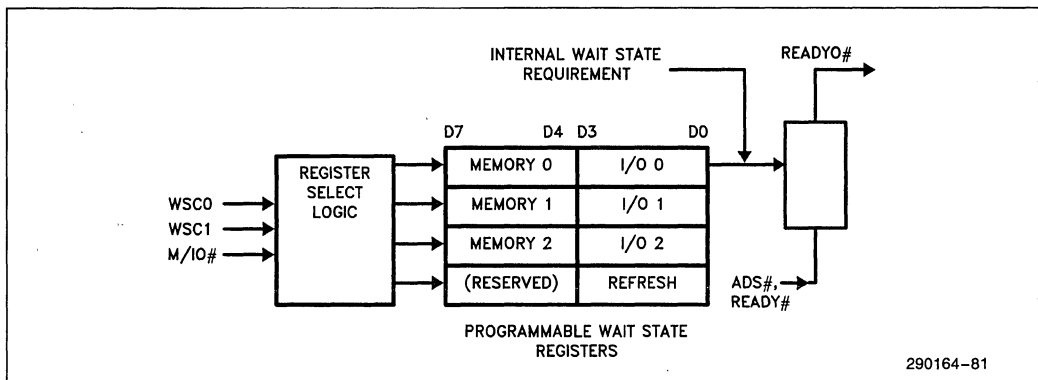


Figure 6-1. Wait State Generator Block Diagram

6.3 Bus Function

6.3.1 WAIT STATES IN NON-PIPELINED CYCLE

The timing diagram of two typical non-pipelined cycles with 82370 generated wait states is shown in Figure 6-2. In this diagram, it is assumed that the internal registers of the 82370 are not addressed. During the first T2 state of each bus cycle, the Wait State Control and the M/IO# inputs are sampled to determine which Wait State Register (if any) is selected. If the WSC inputs are active (i.e. not both are driven HIGH), the pre-programmed number of wait states corresponding to the selected Wait State Register will be requested. This is done by driving the READY# output HIGH during the end of each T2 state.

The WSC (0-1) inputs need only be valid during the very first T2 state of each non-pipelined cycle. As a general rule, the WSC inputs are sampled on the rising edge of the next clock (82384 CLK) after the last state when ADS# (Address Status) is asserted.

The number of wait states generated depends on the type of bus cycle, and the number of wait states requested. The various combinations are discussed below.

1. Access the 82370 internal registers: 2 to 5 wait states, depending upon the specific register addressed. Some back-to-back sequences to the Interrupt Controller will require 7 wait states.

2. Interrupt Acknowledge to the 82370: 5 wait states.

3. Refresh: As programmed in the Refresh Wait State Register (see Register Set Overview). Note that if WCS (0-1) = 11, READY# will stay inactive.

4. Other bus cycles: Depending on WCS (0-1) and M/IO# inputs, these inputs select a Wait State Register in which the number of wait states will be equal to the pre-programmed wait state count in the register plus 1. The Wait State Register selection is defined as follows (Table 6-1).

Table 6-1. Wait State Register Selection

M/IO#	WSC(0-1)	Register Selected
0	00	WAIT REG 0 (I/O half)
0	01	WAIT REG 1 (I/O half)
0	10	WAIT REG 2 (I/O half)
1	00	WAIT REG 0 (MEM half)
1	01	WAIT REG 1 (MEM half)
1	10	WAIT REG 2 (MEM half)
X	11	Wait State Gen. Disabled

The Wait State Control signals, WSC (0-1), can be generated with the address decode and the Read/Write control signals as shown in Figure 6-3.

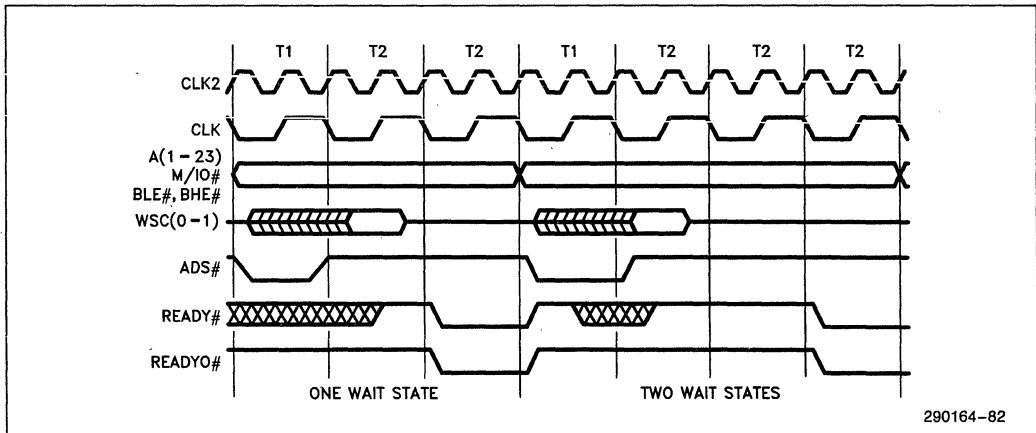


Figure 6-2. Wait States in Non-Pipelined Cycles

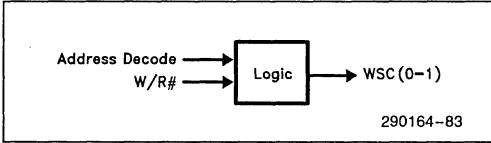


Figure 6-3. WSC (0-1) Generation

Note that during HALT and SHUTDOWN, the number of wait states will depend on the WSC (0-1) inputs, which will select the memory half of one of the Wait State Registers (see CPU Reset and Shutdown Detect).

6.3.2 WAIT STATES IN PIPELINED CYCLES

The timing diagram of two typical pipelined cycles with 82370 generated wait states is shown in Figure 6-4. Again, in this diagram, it is assumed that the 82370 internal registers are not addressed. As defined in the timing of the 80376 processor, the Address (A1-23), Byte Enable (BHE#, BLE#), and other control signals (M/IO#, ADS#) are asserted one T-state earlier than in a non-pipelined cycle; i.e. they are asserted at T2P. Similar to the non-pipelined case, the Wait State Control (WSC) inputs are sampled in the middle of the state after the last state the ADS# signal is asserted. Therefore, the WSC inputs should be asserted during the T1P state of each pipelined cycle (which is one T-state earlier than in the non-pipelined cycle).

The number of wait states generated in a pipelined cycle is selected in a similar manner as in the non-pipelined case discussed in the previous section. The only difference here is that the actual number of wait states generated will be one less than that of the non-pipelined cycle. This is done automatically by the Wait State Generator.

6.3.3 EXTENDING AND EARLY TERMINATING BUS CYCLE

The 82370 allows external logic to either add wait states or cause early termination of a bus cycle by controlling the READY# input to the 82370 and the host processor. A possible configuration is shown in Figure 6-5.

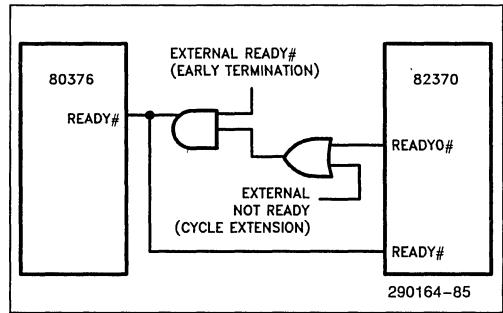


Figure 6-5. External 'READY' Control Logic

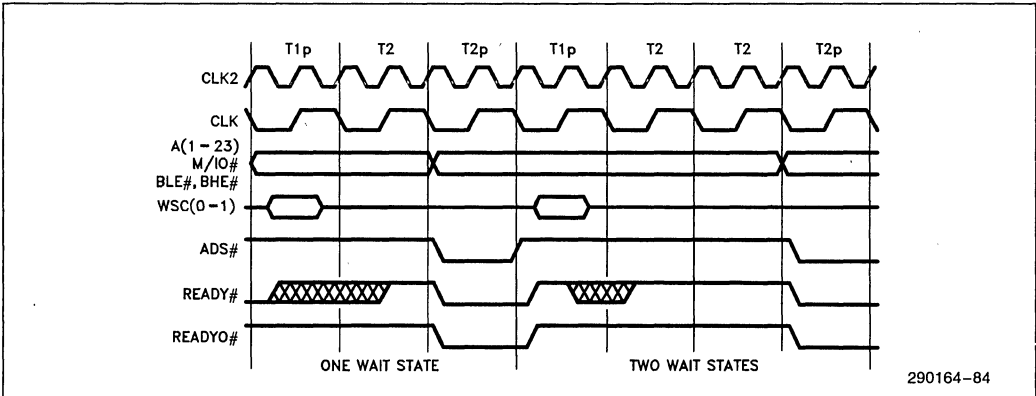


Figure 6-4. Wait States in Pipelined Cycles

The EXT. RDY# (External Ready) signal of Figure 6-5 allows external devices to cause early termination of a bus cycle. When this signal is asserted LOW, the output of the circuit will also go LOW (even though the READY# of the 82370 may still be HIGH). This output is fed to the READY# input of the 80376 and the 82370 to indicate the completion of the current bus cycle.

Similarly, the EXT. NOT READY (External Not Ready) signal is used to delay the READY# input of the processor and the 82370. As long as this signal is driven HIGH, the output of the circuit will drive the READY# input HIGH. This will effectively extend the duration of a bus cycle. However, it is important to

note that if the two-level logic is not fast enough to satisfy the READY# setup time, the OR gate should be eliminated. Instead, the 82370 Wait State Generator can be disabled by driving both WSC (0-1) HIGH. In this case, the addressed memory or I/O device should activate the external READY# input whenever it is ready to terminate the current bus cycle.

Figures 6-6 and 6-7 show the timing relationships of the ready signals for the early termination and extension of the bus cycles. Section 6-7, Application Issues, contains a detailed timing analysis of the external circuit.

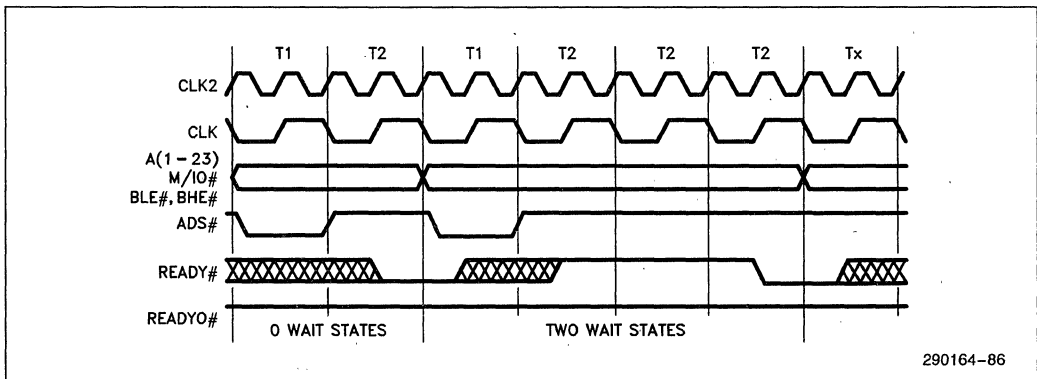


Figure 6-6. Early Termination of Bus Cycle By 'READY #'

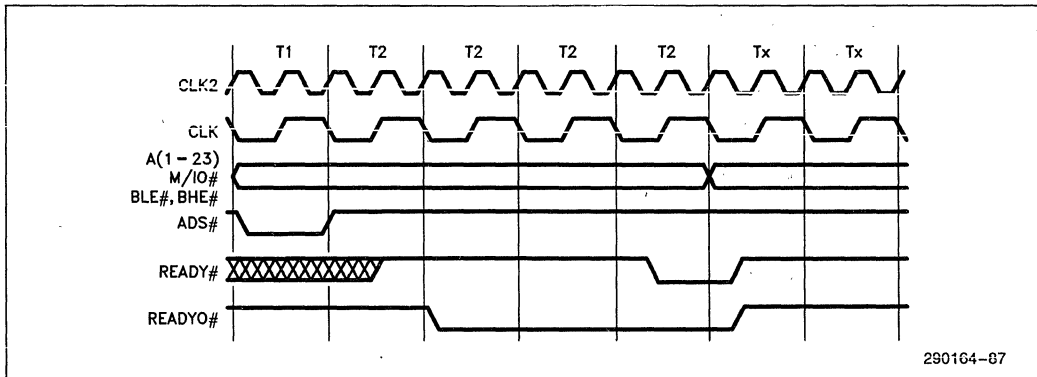


Figure 6-7. Extending Bus Cycle by 'READY #'

Due to the following implications, it should be noted that early termination of bus cycles in which 82370 internal registers are accessed is not recommended.

1. Erroneous data may be read from or written into the addressed register.
2. The 82370 must be allowed to recover either before HLDA (Hold Acknowledge) is asserted or before another bus cycle into an 82370 internal register is initiated.

The recovery time, in clock periods, equals the remaining wait states that were avoided plus 4.

6.4 Register Set Overview

Altogether, there are four 8-bit internal registers associated with the Wait State Generator. The port address map of these registers is shown below in Table 6-2. A detailed description of each follows.

Table 6-2. Register Address Map

Port Address	Description
72H	Wait State Reg 0 (read/write)
73H	Wait State Reg 1 (read/write)
74H	Wait State Reg 2 (read/write)
75H	Ref. Wait State Reg (read/write)

WAIT STATE REGISTER 0, 1, 2

These three 8-bit read/write registers are functionally identical. They are used to store the pre-programmed wait state count. One half of each register contains the wait state count for I/O accesses while the other half contains the count for memory accesses. The total number of wait states generated will depend on the type of bus cycle. For a non-pipelined cycle, the actual number of wait states requested is equal to the wait state count plus 1. For a pipelined cycle, the number of wait states will be equal to the wait state count in the selected register. Therefore, the Wait State Generator is capable of generating 1 to 16 wait states in non-pipelined mode, and 0 to 15 wait states in pipelined mode.

Note that the minimum wait state count in each register is 0. This is equivalent to 0 wait states for a pipelined cycle and 1 wait state for a non-pipelined cycle.

REFRESH WAIT STATE REGISTER

Similar to the Wait State Registers discussed above, this 4-bit register is used to store the number of wait states to be generated during a DRAM refresh cycle.

Note that the Refresh Wait State Register is not selected by the WSC inputs. It will automatically be chosen whenever a DRAM refresh cycle occurs. If the Wait State Generator is disabled during the refresh cycle (WSC (0-1) = 11), READY# will stay inactive and the Refresh Wait State Register is ignored.

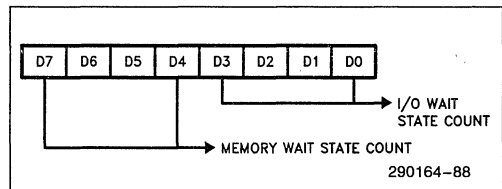
6.5 Programming

Using the Wait State Generator is relatively straightforward. No special programming sequence is required. In order to ensure the expected number of wait states will be generated when a register is selected, the registers to be used must be programmed after power-up by writing the appropriate wait state count into each register. Note that upon hardware reset, all Wait State Registers are initialized with the value FFH, giving the maximum number of wait states possible. Also, each register can be read to check the wait state count previously stored in the register.

6.6 Register Bit Definition

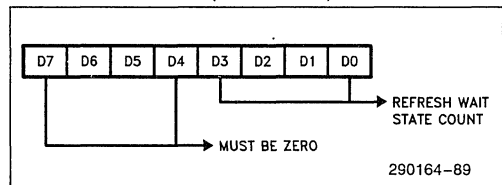
WAIT STATE REGISTER 0, 1, 2

Port Address	Description
72H	Wait State Register 0 (read/write)
73H	Wait State Register 1 (read/write)
74H	Wait State Register 2 (read/write)



REFRESH WAIT STATE REGISTER

Port Address: 75H (Read/Write)



6.7 Application Issues

6.7.1 EXTERNAL 'READY' CONTROL LOGIC

As mentioned in section 6.3.3, wait state cycles generated by the 82370 can be terminated early or extended longer by means of additional external logic (see Figure 6-5). In order to ensure that the READY# input timing requirement of the 80376 and the 82370 is satisfied, special care must be taken when designing this external control logic. This section addresses the design requirements.

A simplified block diagram of the external logic along with the READY# timing diagram is shown in Figure 6-8. The purpose is to determine the maximum delay

time allowed in the external control logic in order to satisfy the READY# setup time.

First, it will be assumed that the 80376 is running at 16 MHz (i.e. CLK2 is 32 MHz). Therefore, one bus state (two CLK2 periods) will be equivalent to 62.5 ns. According to the AC specifications of the 82370, the maximum delay time for valid READY# signal is 31 ns after the rising edge of CLK2 in the beginning of T2 (for non-pipelined cycle) or T2P (for pipelined cycle). Also, the minimum READY# setup time of the 80376 and the 82370 should be 19 ns before the rising edge of CLK2 at the beginning of the next bus state. This limits the total delay time for the external READY# control logic to be 12.5 ns (62.5-31-19) in order to meet the READY# setup timing requirement.

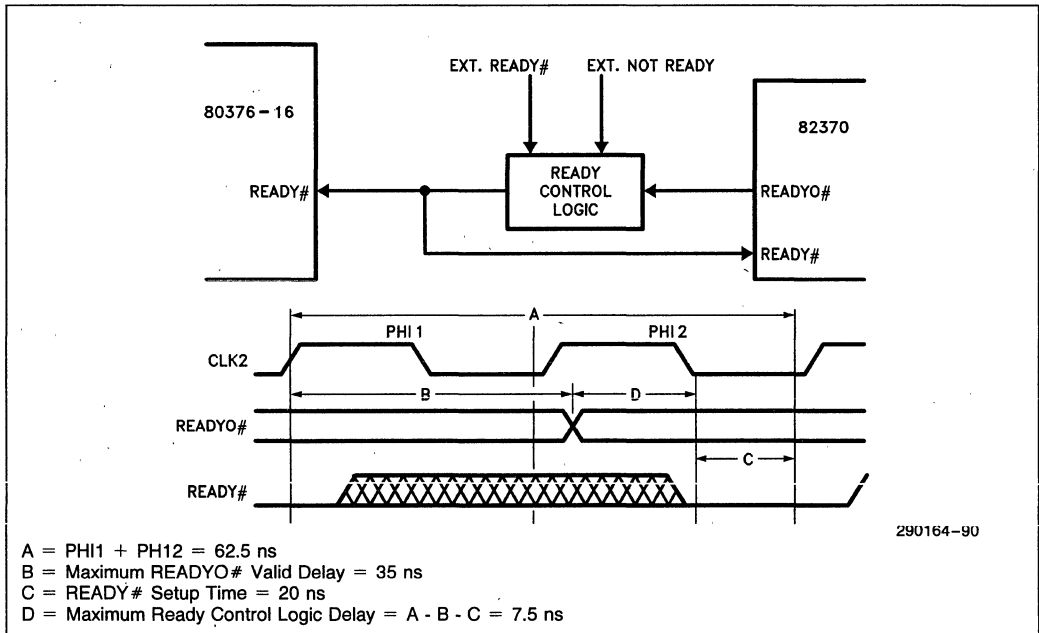


Figure 6-8: 'READY' Timing Consideration

7.0 DRAM REFRESH CONTROLLER

7.1 Functional Description

The 82370 DRAM Refresh Controller consists of a 24-bit Refresh Address Counter and Refresh Request logic for DRAM refresh operations (see Figure 7-1). TIMER 1 can be used as a trigger signal to the DRAM Refresh Request logic. The Refresh Bus Size can be programmed to be 8- or 16-bit wide. Depending on the Refresh Bus Size, the Refresh Address Counter will be incremented with the appropriate value after every refresh cycle. The internal logic of the 82370 will give the Refresh operation the highest priority in the bus control arbitration process. Bus control is not released and re-requested if the 82370 is already a bus master.

7.2 Interface Signals

7.2.1 TOUT1/REF

The dual function output pin of TIMER 1 (TOUT1/REF#) can be programmed to generate DRAM Refresh signal. If this feature is enabled, the rising edge of TIMER 1 output (TOUT1#) will trigger the DRAM Refresh Request logic. After some delay for gaining access of the bus, the 82370 DRAM Controller will generate a DRAM Refresh signal by driving REF# output LOW. This signal is cleared after the refresh cycle has taken place, or by a hardware reset.

If the DRAM Refresh feature is disabled, the TOUT1/REF# output pin is simply the TIMER 1 output. Detailed information of how TIMER 1 operates is discussed in section 6—Programmable Interval Timer, and will not be repeated here.

7.3 Bus Function

7.3.1 ARBITRATION

In order to ensure data integrity of the DRAMs, the 82370 gives the DRAM Refresh signal the highest priority in the arbitration logic. It allows DRAM Refresh to interrupt DMA in progress in order to perform the DRAM Refresh cycle. The DMA service will be resumed after the refresh is done.

In case of a DRAM Refresh during a DMA process, the cascaded device will be requested to get off the bus. This is done by de-asserting the EDACK signal. Once DREQn goes inactive, the 82370 will perform the refresh operation. Note that the DMA controller does not completely relinquish the system bus during refresh. The Refresh Generator simply “steals” a bus cycle between DMA accesses.

Figure 7-2 shows the timing diagram of a Refresh Cycle. Upon expiration of TIMER 1, the 82370 will try to take control of the system bus by asserting HOLD. As soon as the 82370 see HLDA go active, the DRAM Refresh Cycle will be carried out by activating the REF# signal as well as the address and control signals on the system bus (Note that REF# will not be active until two CLK periods HLDA is asserted). The address bus will contain the 24-bit ad-

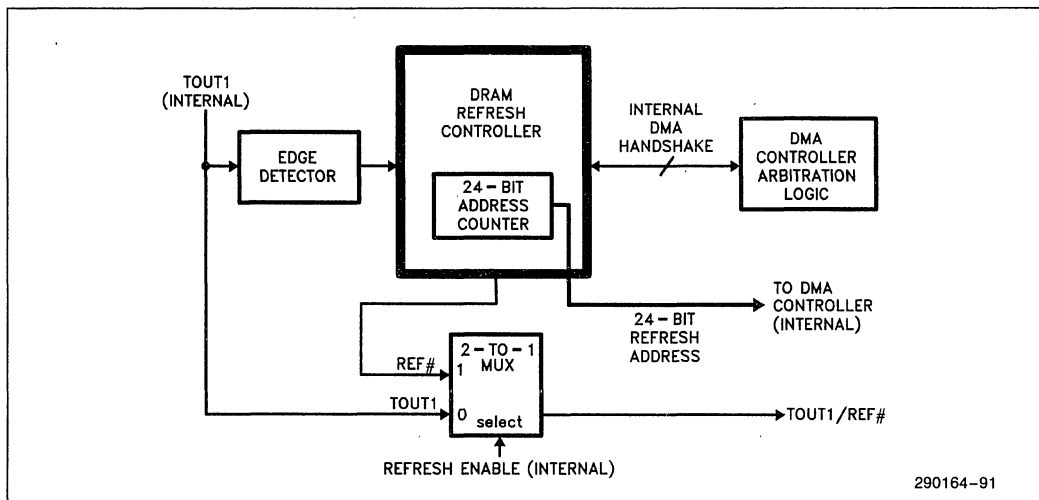


Figure 7-1. DRAM Refresh Controller

dress currently in the Refresh Address Counter. The control signals are driven the same way as in a Memory Read cycle. This "read" operation is complete when the READY# signal is driven LOW. Then, the 82370 will relinquish the bus by de-asserting HOLD. Typically, a Refresh Cycle without wait states will take five bus states to execute. If "n" wait states are added, the Refresh Cycle will last for five plus "n" bus states.

How often the Refresh Generator will initiate a refresh cycle depends on the frequency of CLKIN as well as TIMER 1's programmed mode of operation. For this specific application, TIMER 1 should be programmed to operate in Mode 2 to generate a constant clock rate. See section 6—Programmable Interval Timer for more information on programming the timer. One DRAM Refresh Cycle will be generated each time TIMER 1 expires (when TOUT1 changes from LOW to HIGH).

The Wait State Generator can be used to insert wait states during a refresh cycle. The 82370 will automatically insert the desired number of wait states as programmed in the Refresh Wait State Register (see Wait State Generator).

7.4 Modes of Operation

7.4.1 WORD SIZE AND REFRESH ADDRESS COUNTER

The 82370 supports 8- and 16-bit refresh cycle. The bus width during a refresh cycle is programmable (see Programming). The bus size can be programmed via the Refresh Control Register (see Register Overview). If the DRAM bus size is 8- or 16-bits, the Refresh Address Counter will be incremented by 1 or 2, respectively.

The Refresh Address Counter is cleared by a hardware reset.

7.5 Register Set Overview

The Refresh Generator has two internal registers to control its operation. They are the Refresh Control Register and the Refresh Wait State Register. Their port address map is shown in Table 7-1 below.

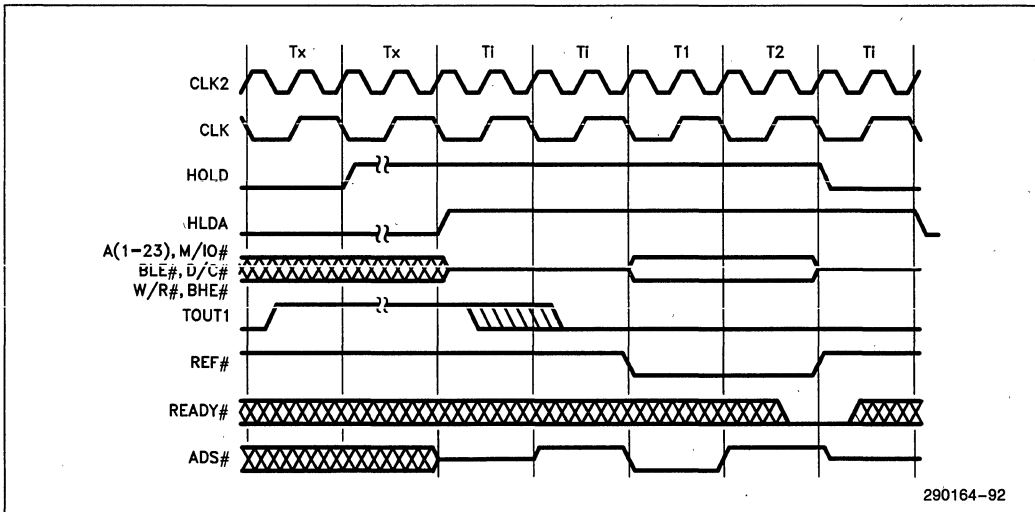


Figure 7-2. 82370 Refresh Cycle

290164-92

Table 7-1. Register Address Map

Port Address	Description
1CH	Refresh Control Reg. (read/write)
75H	Ref. Wait State Reg. (read/write)

The Refresh Wait State Register is not part of the Refresh Generator. It is only used to program the number of wait states to be inserted during a refresh cycle. This register is discussed in detailed in section 7 (Wait State Generator) and will not be repeated here.

REFRESH CONTROL REGISTER

This 2-bit register serves two functions. First, it is used to enable/disable the DRAM Refresh function output. If disabled, the output of TIMER 1 is simply used as a general purpose timer. The second function of this register is to program the DRAM bus size for the refresh operation. The programmed bus size also determines how the Refresh Address Counter will be incremented after each refresh operation.

7.6 Programming

Upon hardware reset, the DRAM Refresh function is disabled (the Refresh Control Register is cleared). The following programming steps are needed before the Refresh Generator can be used. Since the rate of refresh cycles depends on how TIMER 1 is programmed, this timer must be initialized with the desired mode of operation as well as the correct refresh interval (see Programming Interval Timer). Whether or not wait states are to be generated during a refresh cycle, the Refresh Wait State Register must also be programmed with the appropriate value. Then, the DRAM Refresh feature must be enabled and the DRAM bus width should be defined. These can be done in one step by writing the appro-

appropriate control word into the Refresh Control Register (see Register Bit Definition). After these steps are done, the refresh operation will automatically be invoked by the Refresh Generator upon expiration of Timer 1.

In addition to the above programming steps, it should be noted that after reset, although the TOUT1/REF# becomes the Time 1 output, the state of this pin in undefined. This is because the Timer module has not been initialized yet. Therefore, if this output is used as a DRAM Refresh signal, this pin should be disqualified by external logic until the Refresh function is enabled. One simple solution is to logically AND this output with HLDA, since HLDA should not be active after reset.

7.7 Register Bit Definition

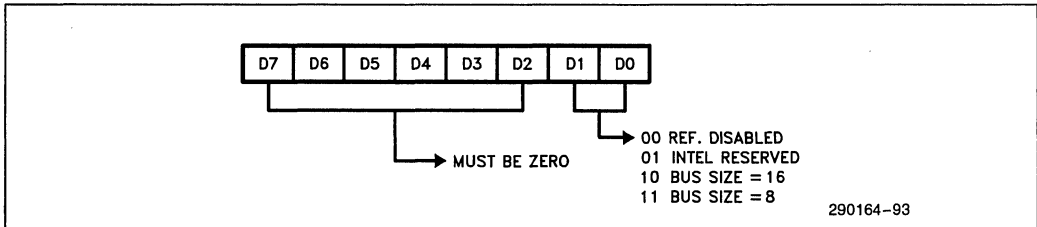
REFRESH CONTROL REGISTER

Port Address: 1CH (Read/Write)

8.0 RELOCATION REGISTER AND ADDRESS DECODE

8.1 Relocation Register

All the integrated peripheral devices in the 82370 are controlled by a set of internal registers. These registers span a total of 256 consecutive address locations (although not all the 256 locations are used). The 82370 provides a Relocation Register which allows the user to map this set of internal registers into either the memory or I/O address space. The function of the Relocation Register is to define the base address of the internal register set of the 82370 as well as if the registers are to be memory- or I/O-mapped. The format of the Relocation Register is depicted in Figure 9-1.



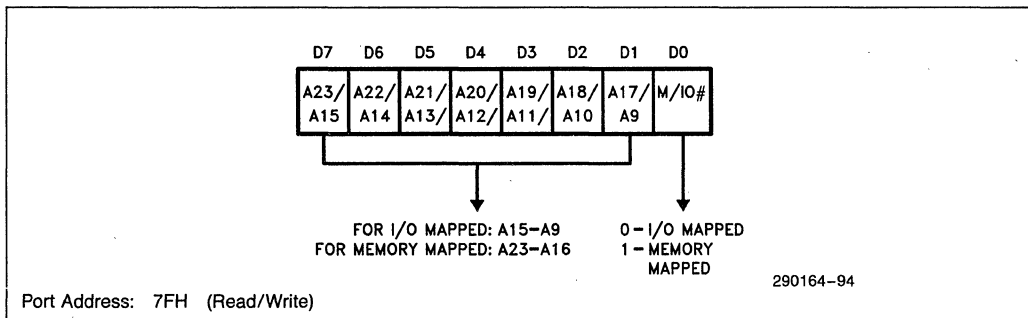


Figure 8-1. Relocation Register

Note that the Relocation Register is part of the internal register set of the 82370. It has a port address of 7FH. Therefore, any time the content of the Relocation Register is changed, the physical location of this register will also be moved. Upon reset of the 82370, the content of the Relocation Register will be cleared. This implies that the 82370 will respond to its I/O addresses in the range of 0000H to 00FFH.

Address signals BHE#, BLE#, A1-A7 will be used to select one of the internal registers to be accessed. Bit 1 to Bit 7 of the Relocation Register will correspond to A17-A23, respectively. A16 is assumed to be '0', and A8-A15 are ignored. Consider the following example.

8.1.1 I/O-MAPPED 82370

As shown in the figure, Bit 0 of the Relocation Register determines whether the 82370 registers are to be memory-mapped or I/O mapped. When Bit 0 is set to '0', the 82370 will respond to I/O Addresses. Address signals BHE#, BLE#, A1-A7 will be used to select one of the internal registers to be accessed. Bit 1 to Bit 7 of the Relocation Register will correspond to A9 to A15 of the Address bus, respectively. Together with A8 implied to be '0', A15 to A8 will be fully decoded by the 82370. The following shows how the 82370 is mapped into the I/O address space.

Example

Relocation Register = 11001110 (0CEH)

82370 will respond to I/O address range from 0CE00H to 0CEFFH.

Therefore, this I/O mapping mechanism allows the 82370 internal registers to be located on any even, contiguous, 256 byte boundary of the system I/O space.

8.1.2 MEMORY-MAPPED 82370

When Bit 0 of the Relocation Register is set to '1', the 82370 will respond to memory addresses. Again,

Example

Relocation Register = 10100111 (0A7H)

The 82370 will respond to memory addresses in the range of A6XX00H to A60XFFFH (where 'X' is don't care).

This scheme implies that the internal registers can be located in any even, contiguous, 2**16 byte page of the memory space.

8.2 Address Decoding

As mentioned previously, the 82370 internal registers do not occupy the entire contiguous 256 address locations. Some of the locations are 'unoccupied'. The 82370 always decodes the lower 8 address signals (BHE#, BLE#, A1-A7) to determine if any one of its registers is being accessed. If the address does not correspond to any of its registers, the 82370 will not respond. This allows external devices to be located within the 'holes' in the 82370 address space. Note that there are several unused addresses reserved for future Intel peripheral devices.

8.3 Chip-Select (CHPSEL #)

The Chip-Select signal (CHPSEL #) will go active when the 82370 is addressed in a Slave bus

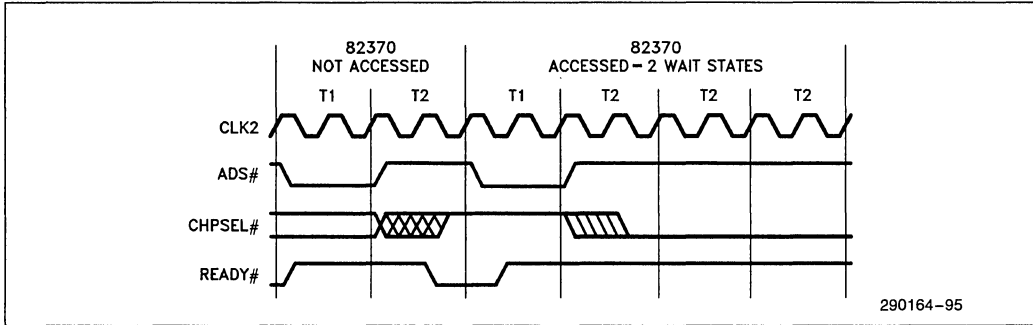


Figure 8-2. CHPSEL# Timing

cycle (either read or write), or in an interrupt acknowledge cycle in which the 82370 will drive the Data Bus. For a given bus cycle, CHPSEL# becomes active and valid in the first T2 (in a non-pipelined cycle) or in T1P (in a pipelined cycle). It will stay valid until the cycle is terminated by READY# driven active. As CHPSEL# becomes valid well before the 82370 drives the Data Bus, it can be used to control the transceivers that connect the local CPU bus to the system bus. The timing diagram of CHPSEL# is shown in Figure 8-2.

9.0 CPU RESET AND SHUTDOWN DETECT

The 82370 will activate the CPURST signal to reset the host processor when one of the following conditions occurs:

- 82370 RESET is active;
- 82370 detects a 80376 Shutdown cycle (this feature can be disabled);
- CPURST software command is issued to 80376.

Whenever the CPURST signal is activated, the 82370 will reset its own internal Slave-Bus state machine.

9.1 Hardware Reset

Following a hardware reset, the 82370 will assert its CPURST output to reset the host processor. This output will stay active for as long as the RESET input is active. During a hardware reset, the 82370 internal registers will be initialized as defined in the corresponding functional descriptions.

9.2 Software Reset

CPURST can be generated by writing the following bit pattern into 82370 register location 64H.

```
D7 . . . . D0
1 1 1 1 X X X 0
```

The Write operation into this port is considered as an 82370 access and the internal Wait State Generator will automatically determine the required number of wait states. The CPURST will be active following the completion of the Write cycle to this port. This signal will last for 62 CLK2 periods. The 82370 should not be accessed until the CPURST is deactivated.

This internal port is Write-Only and the 82370 will not respond to a Read operation to this location. Also, during a software reset command, the 82370 will reset its Slave-Bus state machine. However, its internal registers remain unchanged. This allows the operating system to distinguish a 'warm' reset by reading any 82370 internal register previously programmed for a non-default value. The Diagnostic registers can be used for this purpose (see Internal Control and Diagnostic Ports).

9.3 Shutdown Detect

The 82370 is constantly monitoring the Bus Cycle Definition signals (M/IO#, D/C#, W/R#) and is able to detect when the 80376 is in a Shutdown bus cycle. Upon detection of a processor shutdown, the 82370 will activate the CPURST output for 62 CLK2 periods to reset the host processor. This signal is generated after the Shutdown cycle is terminated by the READY# signal.

Although the 82370 Wait State Generator will not automatically respond to a Shutdown (or Halt) cycle, the Wait State Control inputs (WSC0, WSC1) can be used to determine the number of wait states in the same manner as other non-82370 bus cycles.

This Shutdown Detect feature can be enabled or disabled by writing a control bit in the Internal Control Port at address 61H (see Internal Control and Diagnostic Ports). This feature is disabled upon a hardware reset of the 82370. As in the case of Software Reset, the 82370 will reset its Slave-Bus state machine but will not change any of its internal register contents.

10.0 INTERNAL CONTROL AND DIAGNOSTIC PORTS

10.1 Internal Control Port

The format of the Internal Control Port of the 82370 is shown in Figure 10-1. This Control Port is used to enable/disable the Processor Shutdown Detect mechanism as well as controlling the Gate inputs of the Timer 2 and 3. Note that this is a Write-Only port. Therefore, the 82370 will not respond to a read operation to this port. Upon hardware reset, this port will be cleared; i.e., the Shutdown Detect feature and the Gate inputs of Timer 2 and 3 are disabled.

Port Address: 61H (Write only)

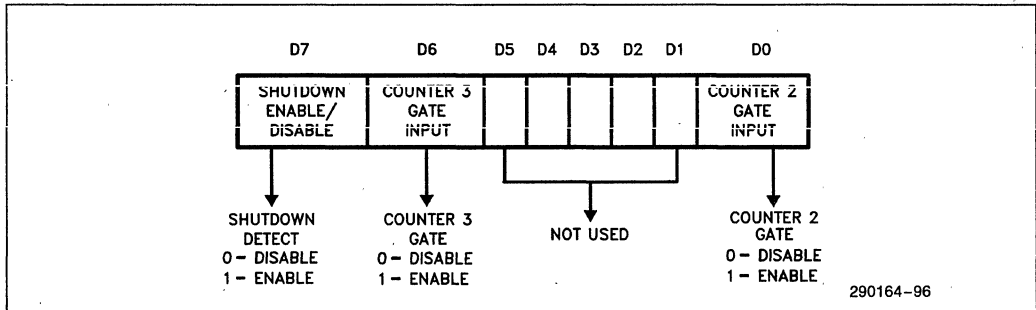


Figure 10-1. Internal Control Port

10.2 Diagnostic Ports

Two 8-bit read/write Diagnostic Ports are provided in the 82370. These are two storage registers and have no effect on the operation of the 82370. They can be used to store checkpoint data or error codes in the power-on sequence and in the diagnostic service routines. As mentioned in the CPU RESET AND SHUTDOWN DETECT section, these Diagnostic Ports can be used to distinguish between 'cold' and 'warm' reset. Upon hardware reset, both Diagnostic Ports are cleared. The address map of these Diagnostic Ports is shown in Figure 10-2.

Port	Address
Diagnostic Port 1 (Read/Write)	80H
Diagnostic Port 2 (Read/Write)	88H

Figure 10-2. Address Map of Diagnostic Ports

11.0 INTEL RESERVED I/O PORTS

There are nineteen I/O ports in the 82370 address space which are reserved for Intel future peripheral device use only. Their address locations are: 10H, 12H, 14H, 16H, 2AH, 3DH, 3EH, 45H, 46H, 76H, 77H, 7DH, 7EH, CCH, CDH, D0H, D2H, D4H, and D6H. These addresses should not be used in the system since the 82370 will respond to read/write operations to these locations and bus contention may occur if any peripheral is assigned to the same address location.

12.0 PACKAGE THERMAL SPECIFICATIONS

The intel 82370 Integrated System Peripheral is specified for operation when case temperature is within the range of 0°C to 78°C for the ceramic 132-pin PGA package, and 68°C for the 100-pin plastic package. The case temperature may be measured in any environment, to determine whether the 82370 is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature is guaranteed as long as T_c is not violated. The ambient temperature can be

calculated from the θ_{jc} and θ_{ja} from the following equations:

$$T_J = T_c + P \cdot \theta_{jc}$$

$$T_A = T_j - P \cdot \theta_{ja}$$

$$T_C = T_a + P \cdot [\theta_{ja} - \theta_{jc}]$$

Values for θ_{ja} and θ_{jc} are given in Table 12.1 for the 100-lead fine pitch. θ_{ja} is given at various airflows. Table 12.2 shows the maximum T_a allowable (without exceeding T_c) at various airflows. Note that T_a can be improved further by attaching "fins" or a "heat sink" to the package. P is calculated using the maximum *hot* I_{CC} .

Table 12.1 82370 Package Thermal Characteristics
Thermal Resistances (°C/Watt) θ_{jc} and θ_{ja}

Package	θ_{jc}	θ_{ja} Versus Airflow-ft ³ /min (m ³ /sec)					
		0	200	400	600	800	1000
		(0)	(1.01)	(2.03)	(3.04)	(4.06)	(5.07)
100L Fine Pitch	7	33	27	24	21	18	17
132L PGA	2	21	17	14	12	11	10

Table 12.2 82370 Maximum Allowable Ambient Temperature at Various Airflows

Package	θ_{jc}	$T_a(c)$ Versus Airflow-ft ³ /min (m ³ /sec)					
		0	200	400	600	800	1000
		(0)	(1.01)	(2.03)	(3.04)	(4.06)	(5.07)
100L Fine Pitch	7	63	74	79	85	91	92
132L PGA	2	74	83	88	93	97	99

100L PQFP Pkg:
 $T_c = T_a + P \cdot (\theta_{ja} - \theta_{jc})$
 $T_c = 63 + 1.21(33 - 7)$
 $T_c = 63 + 1.21(26)$
 $T_c = 63 + 31.46$
 $T_c = 94^\circ\text{C}$

132L PGA Pkg:
 $T_c = T_a + P \cdot (\theta_{ja} - \theta_{jc})$
 $T_c = 74 + 1.21(21 - 2)$
 $T_c = 74 + 1.21(19)$
 $T_c = 74 + 22.99$
 $T_c = 96^\circ\text{C}$

13.0 ELECTRICAL SPECIFICATIONS

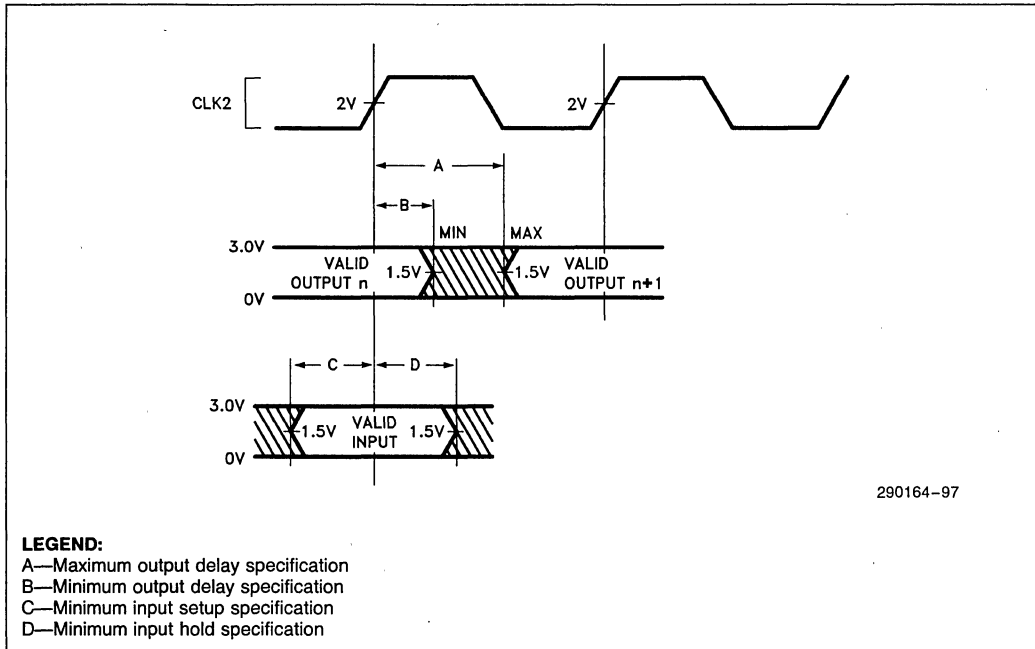
82370 D.C. Specifications Functional Operating Range:

$V_{CC} = 5.0V \pm 10\%$; $T_{CASE} = 0^{\circ}C$ to $96^{\circ}C$ for 132-pin PGA, $0^{\circ}C$ to $94^{\circ}C$ for 100-pin plastic

Symbol	Parameter Description	Min	Max	Units	Notes
V_{IL}	Input Low Voltage	-0.3	0.8	V	(Note 1)
V_{IH}	Input High Voltage	2.0	$V_{CC} + 0.3$	V	
V_{ILC}	CLK2 Input Low Voltage	-0.3	0.8	V	(Note 1)
V_{IHC}	CLK2 Input High Voltage	$V_{CC} - 0.8$	$V_{CC} + 0.3$	V	
V_{OL}	Output Low Voltage $I_{OL} = 4$ mA: A ₁₋₂₃ , D ₀₋₁₅ , BHE#, BLE# $I_{OL} = 5$ mA: All Others		0.45 0.45	V V	
V_{OH}	Output High Voltage				
$I_{OH} = -1$ mA	A _{23-A1} , D _{15-D0} , BHE#, BLE#	2.4		V	(Note 5)
$I_{OH} = -0.2$ mA	A _{23-A1} , D _{15-D0} , BHE#, BLE#	$V_{CC} - 0.5$		V	(Note 5)
$I_{OH} = -0.9$ mA	All Others	2.4		V	(Note 5)
$I_{OH} = -0.18$ mA	All Others	$V_{CC} - 0.5$		V	(Note 5)
I_{LI}	Input Leakage Current All Inputs Except: IRQ11# - IRQ23# EOP#, TOUT2/IRQ3# DREQ4/IRQ9#		± 15	μA	
I_{LI1}	Input Leakage Current Inputs: IRQ11# - IRQ23# EOP#, TOUT2/IRQ3# DREQ4/IRQ9#	10	-300	μA	$0 < V_{IN} < V_{CC}$ (Note 3)
I_{LO}	Output Leakage Current		± 15	μA	$0 < V_{IN} < V_{CC}$
I_{CC}	Supply Current (CLK2 = 32 MHz)		220	mA	(Note 4)
C_I	Input Capacitance		12	pF	(Note 2)
C_{CLK}	CLK2 Input Capacitance		20	pF	(Note 2)

NOTES:

1. Minimum value is not 100% tested.
2. $f_C = 1$ MHz; sampled only.
3. These pins have weak internal pullups. They should not be left floating.
4. I_{CC} is specified with inputs driven to CMOS levels, and outputs driving CMOS loads. I_{CC} may be higher if inputs are driven to TTL levels, or if outputs are driving TTL loads.
5. Tested at the minimum operating frequency of the part.



290164-97

Figure 13-1. Drive Levels and Measurement Points for A.C. Specification

82370 A.C. Specifications These A.C. timings are tested at 1.5V thresholds, except as noted.
 Functional Operating Range: $V_{CC} = 5.0V \pm 10\%$; $T_{CASE} = 0^{\circ}C$ to $96^{\circ}C$ for 132-pin PGA, $0^{\circ}C$ to $94^{\circ}C$ for 100-pin plastic

Symbol	Parameter Description	Min	Max	Units	Notes
	Operating Frequency $1/(t1a \times 2)$	4	16	MHz	
t1	CLK2 Period	31	125	ns	
t2a	CLK2 High Time	9		ns	At 2.0V
t2b	CLK2 High Time	5		ns	At $V_{CC} - 0.8V$
t3a	CLK2 Low Time	9		ns	At 2.0V
t3b	CLK2 Low Time	7		ns	At 0.8V
t4	CLK2 Fall Time		7	ns	$V_{CC} - 0.8V$ to 0.8V
t5	CLK2 Rise Time		7	ns	0.8V to $V_{CC} - 0.8V$
t6	A1-A23, BHE#, BLE# EDACK0-EDACK2 Valid Delay	4	36	ns	$C_L = 120$ pF
t7	A1-A23, BHE#, BLE# EDACK0-EDACK3 Float Delay	4	40	ns	(Note 1)
t8	A1-A23, BHE#, BLE# Setup Time	6		ns	
t9	A1-A23, BHE#, BLE# Hold Time	4		ns	
t10	W/R#, M/IO#, D/C# Valid Delay	4	33	ns	$C_L = 75$ pF
t11	W/R#, M/IO#, D/C# Float Delay	4	35	ns	(Note 1)

82370 A.C. Specifications These A.C. timings are tested at 1.5V thresholds, except as noted.
 Functional Operating Range: $V_{CC} = 5.0V \pm 10\%$; $T_{CASE} = 0^{\circ}C$ to $96^{\circ}C$ for 132-pin PGA, $0^{\circ}C$ to $94^{\circ}C$ for 100-pin plastic (Continued)

Symbol	Parameter Description	Min	Max	Units	Notes
t12	W/R#, M/IO#, D/C# Setup Time	6		ns	
t13	W/R#, M/IO#, D/C# Hold Time	4		ns	
t14	ADS# Valid Delay	6	33	ns	CL = 50 pF (Note 1)
t15	ADS# Float Delay	4	35	ns	
t16	ADS# Setup Time	21		ns	
t17	ADS# Hold Time	4		ns	
t18	Slave Mode D0–D15 Read Valid	3	46	ns	CL = 120 pF (Note 1)
t19	Slave Mode D0–D15 Read Float	6	35	ns	
t20	Slave Mode D0–D15 Write Setup	31		ns	
t21	Slave Mode D0–D15 Write Hold	26		ns	
t22	Master Mode D0–D15 Write Valid	4	40	ns	CL = 120 pF (Note 1)
t23	Master Mode D0–D15 Write Float	4	35	ns	
t24	Master Mode D0–D15 Read Setup	8		ns	
t25	Master Mode D0–D15 Read Hold	6		ns	
t26	READY# Setup Time	19		ns	
t27	READY# Hold Time	4		ns	
t28	WSC0–WSC1 Setup Time	6		ns	
t29	WSC0–WSC1 Hold Time	21		ns	
t30	RESET Setup Time	13		ns	
t31	RESET Hold Time	4		ns	
t32	READYO# Valid Delay	4	31	ns	CL = 25 pF
t33	CPURST Valid Delay (Falling Edge Only)	2	18	ns	CL = 50 pF
t34	HOLD Valid Delay	5	33	ns	CL = 100 pF
t35	HLDA Setup Time	21		ns	
t36	HLDA Hold Time	6		ns	
t37a	EOP# Setup (Synchronous)	21		ns	
t38a	EOP# Hold (Synchronous)	6		ns	
t37b	EOP# Setup (Asynchronous)	11		ns	
t38b	EOP# Hold (Asynchronous)	11		ns	
t39	EOP# Valid Delay (Falling Edge Only)	5	38	ns	CL = 100 pF (Note 1)
t40	EOP# Float Delay	5	40	ns	
t41a	DREQ Setup (Synchronous)	21		ns	
t42a	DREQ Hold (Synchronous)	4		ns	
t41b	DREQ Setup (Asynchronous)	11		ns	
t42b	DREQ Hold (Asynchronous)	11		ns	
t43	INT Valid Delay from IRQn		500	ns	
t44	NA# Setup Time	5		ns	
t45	NA# Hold Time	15		ns	

82370 A.C. Specifications These A.C. timings are tested at 1.5V thresholds, except as noted. Functional Operating Range: $V_{CC} = 5.0V \pm 10\%$; $T_{CASE} = 0^{\circ}C$ to $96^{\circ}C$ for 132-pin PGA, $0^{\circ}C$ to $94^{\circ}C$ for 100-pin plastic (Continued)

Symbol	Parameter Description	Min	Max	Units	Notes
t46	CLKIN Frequency	DC	10	MHz	
t47	CLKIN High Time	30		ns	2.0V
t48	CLKIN Low Time	50		ns	0.8V
t49	CLKIN Rise Time		10	ns	0.8V to 3.7V
t50	CLKIN Fall Time		10	ns	3.7V to 0.8V
t51	TOUT1 # /REF# Valid Delay from CLK2 (Refresh)	4	36	ns	$C_L = 120$ pF
t52	TOUT1 # /REF# Valid Delay from CLKIN (Timer)	3	93	ns	$C_L = 120$ pF
t53	TOUT2 # Valid Delay (from CLKIN, Falling Edge Only)	3	93	ns	$C_L = 120$ pF
t54	TOUT2 # Float Delay	3	36	ns	(Note 1)
t55	TOUT3 # Valid Delay (from CLKIN)	3	93	ns	$C_L = 120$ pF
t56	CHPSEL # Valid Delay	1	35	ns	$C_L = 25$ pF

NOTE:

1. Float condition occurs when the maximum output current becomes less than I_{LO} in magnitude. Float delay is not tested. For testing purposes, the float condition occurs when the dynamic output driven voltage changes with current loads.

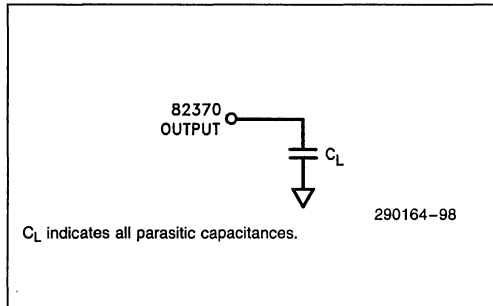


Figure 13-2. A.C. Test Load

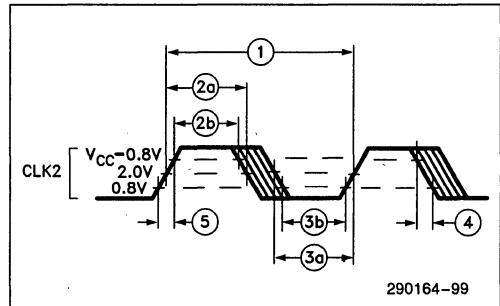


Figure 13-3

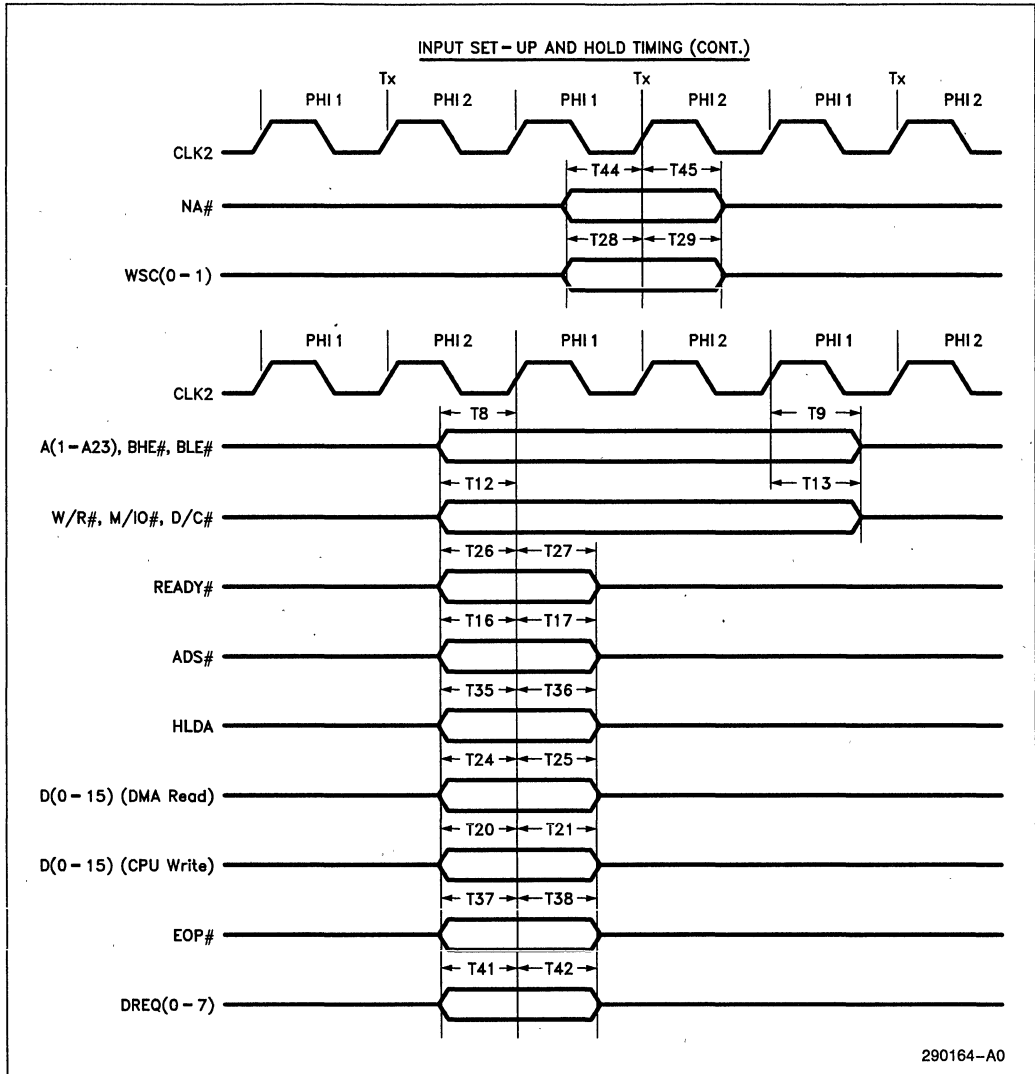
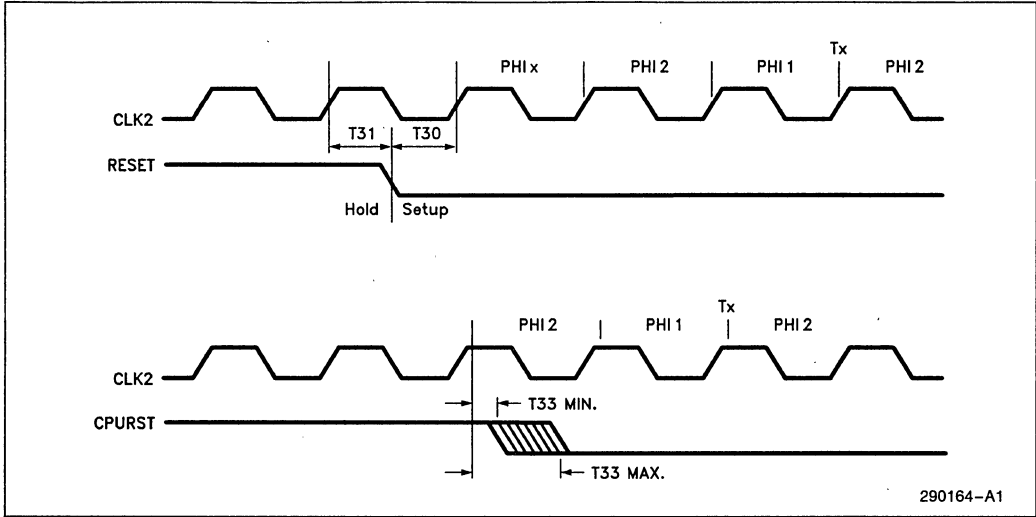
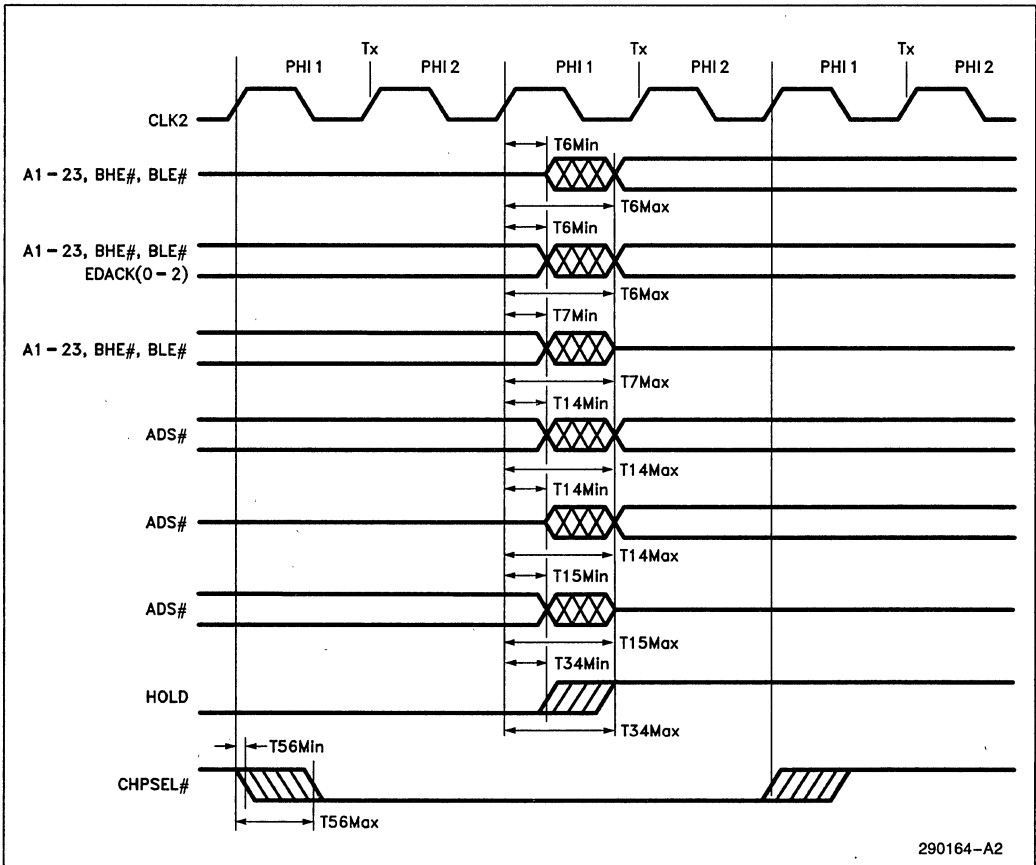


Figure 13-4. Input Setup and Hold Timing



290164-A1

Figure 13-5. Reset Timing



290164-A2

Figure 13-6. Address Output Delays

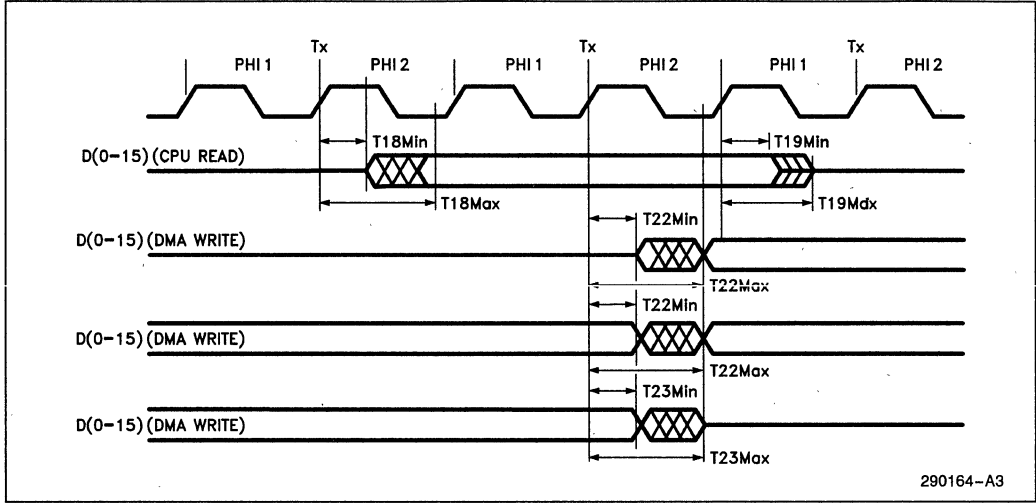


Figure 13-7. Data Bus Output Delays

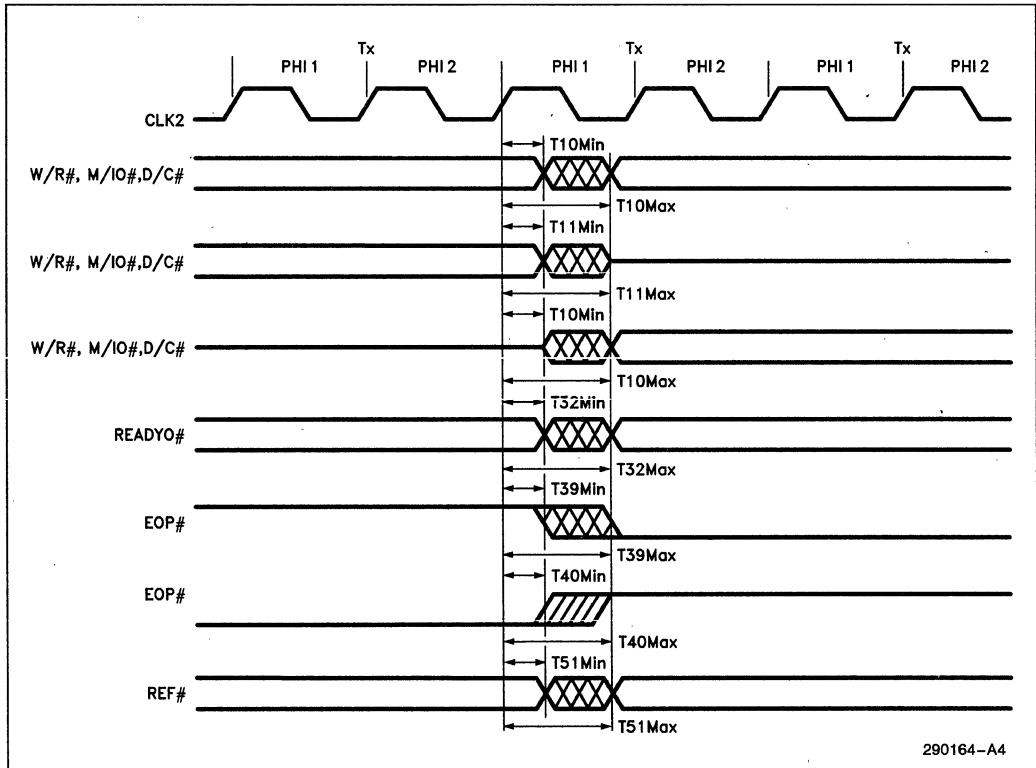
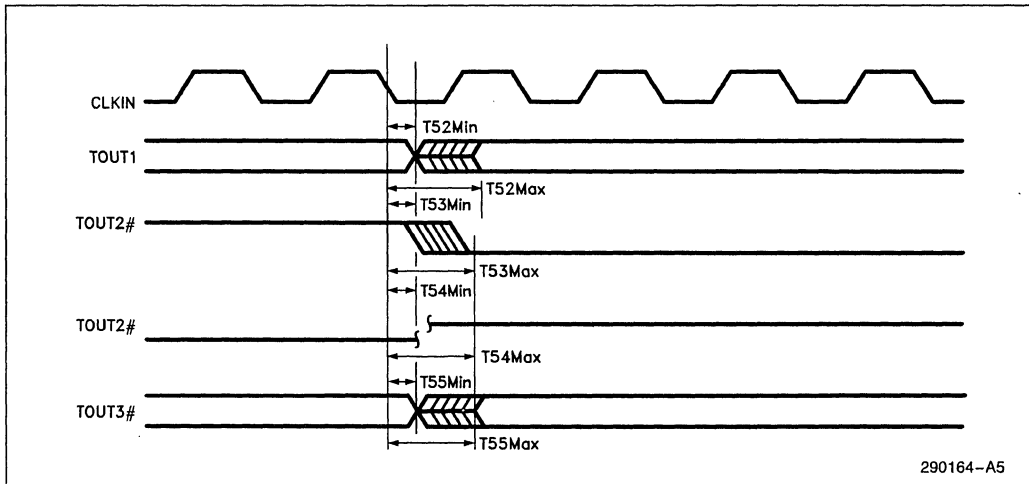


Figure 13-8. Control Output Delays



290164-A5

Figure 13-9. Timer Output Delays

APPENDIX A PORTS LISTED BY ADDRESS

Port Address (HEX)	Description
00	Read/Write DMA Channel 0 Target Address, A0–A15
01	Read/Write DMA Channel 0 Byte Count, B0–B15
02	Read/Write DMA Channel 1 Target Address, A0–A15
03	Read/Write DMA Channel 1 Byte Count, B0–B15
04	Read/Write DMA Channel 2 Target Address, A0–A15
05	Read/Write DMA Channel 2 Byte Count, B0–B15
06	Read/Write DMA Channel 3 Target Address, A0–A15
07	Read/Write DMA Channel 3 Byte Count, B0–B15
08	Read/Write DMA Channel 0–3 Status/Command I Register
09	Read/Write DMA Channel 0–3 Software Request Register
0A	Write DMA Channel 0–3 Set-Reset Mask Register
0B	Write DMA Channel 0–3 Mode Register I
0C	Write Clear Byte-Pointer FF
0D	Write DMA Master-Clear
0E	Write DMA Channel 0–3 Clear Mask Register
0F	Read/Write DMA Channel 0–3 Mask Register
10	Intel Reserved
11	Read/Write DMA Channel 0 Byte Count, B16–B23
12	Intel Reserved
13	Read/Write DMA Channel 1 Byte Count, B16–B23
14	Intel Reserved
15	Read/Write DMA Channel 2 Byte Count, B16–B23
16	Intel Reserved
17	Read/Write DMA Channel 3 Byte Count, B16–B23
18	Write DMA Channel 0–3 Bus Size Register
19	Read/Write DMA Channel 0–3 Chaining Register
1A	Write DMA Channel 0–3 Command Register II
1B	Write DMA Channel 0–3 Mode Register II
1C	Read/Write Refresh Control Register
1E	Reset Software Request Interrupt
20	Write Bank B ICW1, OCW2 or OCW3
	Read Bank B Poll, Interrupt Request or In-Service Status Register
21	Write Bank B ICW2, ICW3, ICW4 or OCW1
	Read Bank B Interrupt Mask Register
22	Read Bank B ICW2
28	Read/Write IRQ8 Vector Register
29	Read/Write IRQ9 Vector Register
2A	Reserved

Port Address (HEX)	Description
2B	Read/Write IRQ11 Vector Register
2C	Read/Write IRQ12 Vector Register
2D	Read/Write IRQ13 Vector Register
2E	Read/Write IRQ14 Vector Register
2F	Read/Write IRQ15 Vector Register
30	Write Bank A ICW1, OCW2 or OCW3 Read Bank A Poll, Interrupt Request or In-Service Status Register
31	Write Bank A ICW2, ICW3, ICW4 or OCW1 Read Bank A Interrupt Mask Register
32	Read Bank A ICW2
38	Read/Write IRQ0 Vector Register
39	Read/Write IRQ1 Vector Register
3A	Read/Write IRQ1.5 Vector Register
3B	Read/Write IRQ3 Vector Register
3C	Read/Write IRQ4 Vector Register
3D	Reserved
3E	Reserved
3F	Read/Write IRQ7 Vector Register
40	Read/Write Counter 0 Register
41	Read/Write Counter 1 Register
42	Read/Write Counter 2 Register
43	Write Control Word Register I—Counter 0, 1, 2
44	Read/Write Counter 3 Register
45	Reserved
46	Reserved
47	Write Word Register II—Counter 3
61	Write Internal Control Port
64	Write CPU Reset Register (Data—1111XXX0H)
72	Read/Write Wait State Register 0
73	Read/Write Wait State Register 1
74	Read/Write Wait State Register 2
75	Read/Write Refresh Wait State Register
76	Reserved
77	Reserved
7D	Reserved
7E	Reserved
7F	Read/Write Relocation Register
80	Read/Write Internal Diagnostic Port 0
81	Read/Write DMA Channel 2 Target Address, A16–A23
82	Read/Write DMA Channel 3 Target Address, A16–A23
83	Read/Write DMA Channel 1 Target Address, A16–A23
87	Read/Write DMA Channel 0 Target Address, A16–A23
88	Read/Write Internal Diagnostic Port 1
89	Read/Write DMA Channel 6 Target Address, A16–A23
8A	Read/Write DMA Channel 7 Target Address, A16–A23
8B	Read/Write DMA Channel 5 Target Address, A16–A23
8F	Read/Write DMA Channel 4 Target Address, A16–A23

Port Address (HEX)	Description
90	Read/Write DMA Channel 0 Requester Address, A0–A15
91	Read/Write DMA Channel 0 Requester Address, A16–A23
92	Read/Write DMA Channel 1 Requester Address, A0–A15
93	Read/Write DMA Channel 1 Requester Address, A16–A23
94	Read/Write DMA Channel 2 Requester Address, A0–A15
95	Read/Write DMA Channel 2 Requester Address, A16–A23
96	Read/Write DMA Channel 3 Requester Address, A0–A15
97	Read/Write DMA Channel 3 Requester Address, A16–A23
98	Read/Write DMA Channel 4 Requester Address, A0–A15
99	Read/Write DMA Channel 4 Requester Address, A16–A23
9A	Read/Write DMA Channel 5 Requester Address, A0–A15
9B	Read/Write DMA Channel 5 Requester Address, A16–A23
9C	Read/Write DMA Channel 6 Requester Address, A0–A15
9D	Read/Write DMA Channel 6 Requester Address, A16–A23
9E	Read/Write DMA Channel 7 Requester Address, A0–A15
9F	Read/Write DMA Channel 7 Requester Address, A16–A23
A0	Write Bank C ICW1, OCW2 or OCW3 Read Bank C Poll, Interrupt Request or In-Service Status Register
A1	Write Bank C ICW2, ICW3, ICW4 or OCW1 Read Bank C Interrupt Mask Register
A2	Read Bank C ICW2
A8	Read/Write IRQ16 Vector Register
A9	Read/Write IRQ17 Vector Register
AA	Read/Write IRQ18 Vector Register
AB	Read/Write IRQ19 Vector Register
AC	Read/Write IRQ20 Vector Register
AD	Read/Write IRQ21 Vector Register
AE	Read/Write IRQ22 Vector Register
AF	Read/Write IRQ23 Vector Register
C0	Read/Write DMA Channel 4 Target Address, A0–A15
C1	Read/Write DMA Channel 4 Byte Count, B0–B15
C2	Read/Write DMA Channel 5 Target Address, A0–A15
C3	Read/Write DMA Channel 5 Byte Count, B0–B15
C4	Read/Write DMA Channel 6 Target Address, A0–A15
C5	Read/Write DMA Channel 6 Byte Count, B0–B15
C6	Read/Write DMA Channel 7 Target Address, A0–A15
C7	Read/Write DMA Channel 7 Byte Count, B0–B15
C8	Read DMA Channel 4–7 Status/Command I Register
C9	Read/Write DMA Channel 4–7 Software Request Register
CA	Write DMA Channel 4–7 Set-Reset Mask Register
CB	Write DMA Channel 4–7 Mode Register I
CC	Reserved
CD	Reserved
CE	Write DMA Channel 4–7 Clear Mask Register
CF	Read/Write DMA Channel 4–7 Mask Register
D0	Intel Reserved
D1	Read/Write DMA Channel 4 Byte Count, B16–B23
D2	Intel Reserved
D3	Read/Write DMA Channel 5 Byte Count, B16–B23

Port Address (HEX)	Description
D4	Intel Reserved
D5	Read/Write DMA Channel 6 Byte Count, B16-B23
D6	Intel Reserved
D7	Read/Write DMA Channel 7 Byte Count, B16-B23
D8	Write DMA Channel 4-7 Bus Size Register
D9	Read/Write DMA Channel 4-7 Chaining Register
DA	Write DMA Channel 4-7 Command Register II
DB	Write DMA Channel 4-7 Mode Register II

APPENDIX B PORTS LISTED BY FUNCTION

Port Address (HEX)	Description
DMA CONTROLLER	
0D	Write DMA Master-Clear
0C	Write DMA Clear Byte-Pointer FF
08	Read/Write DMA Channel 0–3 Status/Command I Register
C8	Read/Write DMA Channel 4–7 Status/Command I Register
1A	Write DMA Channel 0–3 Command Register II
DA	Write DMA Channel 4–7 Command Register II
0B	Write DMA Channel 0–3 Mode Register I
CB	Write DMA Channel 4–7 Mode Register I
1B	Write DMA Channel 0–3 Mode Register II
DB	Write DMA Channel 4–7 Mode Register II
09	Read/Write DMA Channel 0–3 Software Request Register
C9	Read/Write DMA Channel 4–7 Software Request Register
1E	Reset Software Request Interrupt
0E	Write DMA Channel 0–3 Clear Mask Register
CE	Write DMA Channel 4–7 Clear Mask Register
0F	Read/Write DMA Channel 0–3 Mask Register
CF	Read/Write DMA Channel 4–7 Mask Register
0A	Write DMA Channel 0–3 Set-Reset Mask Register
CA	Write DMA Channel 4–7 Set-Reset Mask Register
18	Write DMA Channel 0–3 Bus Size Register
D8	Write DMA Channel 4–7 Bus Size Register
19	Read/Write DMA Channel 0–3 Chaining Register
D9	Read/Write DMA Channel 4–7 Chaining Register
00	Read/Write DMA Channel 0 Target Address, A0–A15
87	Read/Write DMA Channel 0 Target Address, A16–A23
01	Read/Write DMA Channel 0 Byte Count, B0–B15
11	Read/Write DMA Channel 0 Byte Count, B16–B23
90	Read/Write DMA Channel 0 Requester Address, A0–A15
91	Read/Write DMA Channel 0 Requester Address, A16–A23

Port Address (HEX)	Description
DMA CONTROLLER (Continued)	
02	Read/Write DMA Channel 1 Target Address, A0–A15
83	Read/Write DMA Channel 1 Target Address, A16–A23
03	Read/Write DMA Channel 1 Byte Count, B0–B15
13	Read/Write DMA Channel 1 Byte Count, B16–B23
92	Read/Write DMA Channel 1 Requester Address, A0–A15
93	Read/Write DMA Channel 1 Requester Address, A16–A23
04	Read/Write DMA Channel 2 Target Address, A0–A15
81	Read/Write DMA Channel 2 Target Address, A16–A23
05	Read/Write DMA Channel 2 Byte Count, B0–B15
15	Read/Write DMA Channel 2 Byte Count, B16–B23
94	Read/Write DMA Channel 2 Requester Address, A0–A15
95	Read/Write DMA Channel 2 Requester Address, A16–A23
06	Read/Write DMA Channel 3 Target Address, A0–A15
82	Read/Write DMA Channel 3 Target Address, A16–A23
07	Read/Write DMA Channel 3 Byte Count, B0–B15
17	Read/Write DMA Channel 3 Byte Count, B16–B23
96	Read/Write DMA Channel 3 Requester Address, A0–A15
97	Read/Write DMA Channel 3 Requester Address, A16–A23
C0	Read/Write DMA Channel 4 Target Address, A0–A15
8F	Read/Write DMA Channel 4 Target Address, A16–A23
C1	Read/Write DMA Channel 4 Byte Count, B0–B15
D1	Read/Write DMA Channel 4 Byte Count, B16–B23
98	Read/Write DMA Channel 4 Requester Address, A0–A15
99	Read/Write DMA Channel 4 Requester Address, A16–A23
C2	Read/Write DMA Channel 5 Target Address, A0–A15
8B	Read/Write DMA Channel 5 Target Address, A16–A23
C3	Read/Write DMA Channel 5 Byte Count, B0–B15
D3	Read/Write DMA Channel 5 Byte Count, B16–B23
9A	Read/Write DMA Channel 5 Requester Address, A0–A15
9B	Read/Write DMA Channel 5 Requester Address, A16–A23
C4	Read/Write DMA Channel 6 Target Address, A0–A15
89	Read/Write DMA Channel 6 Target Address, A16–A23
C5	Read/Write DMA Channel 6 Byte Count, B0–B15
D5	Read/Write DMA Channel 6 Byte Count, B16–B23
9C	Read/Write DMA Channel 6 Requester Address, A0–A15
9D	Read/Write DMA Channel 6 Requester Address, A16–A23
C6	Read/Write DMA Channel 7 Target Address, A0–A15
8A	Read/Write DMA Channel 7 Target Address, A16–A23
C7	Read/Write DMA Channel 7 Byte Count, B0–B15
D7	Read/Write DMA Channel 7 Byte Count, B16–B23
9E	Read/Write DMA Channel 7 Requester Address, A0–A15
9F	Read/Write DMA Channel 7 Requester Address, A16–A23

Port Address (HEX)	Description
INTERRUPT CONTROLLER	
20	Write Bank B ICW1, OCW2 or OCW3 Read Bank B Poll, Interrupt Request or In-Service Status Register
21	Write Bank B ICW2, ICW3, ICW4 or OCW1 Read Bank B Interrupt Mask Register
22	Read Bank B ICW2
28	Read/Write IRQ8 Vector Register
29	Read/Write IRQ9 Vector Register
2A	Reserved
2B	Read/Write IRQ11 Vector Register
2C	Read/Write IRQ12 Vector Register
2D	Read/Write IRQ13 Vector Register
2E	Read/Write IRQ14 Vector Register
2F	Read/Write IRQ15 Vector Register
A0	Write Bank C ICW1, OCW2 or OCW3 Read Bank C Poll, Interrupt Request or In-Service Status Register
A1	Write Bank C ICW2, ICW3, ICW4 or OCW1 Read Bank C Interrupt Mask Register
A2	Read Bank C ICW2
A8	Read/Write IRQ16 Vector Register
A9	Read/Write IRQ17 Vector Register
AA	Read/Write IRQ18 Vector Register
AB	Read/Write IRQ19 Vector Register
AC	Read/Write IRQ20 Vector Register
AD	Read/Write IRQ21 Vector Register
AE	Read/Write IRQ22 Vector Register
AF	Read/Write IRQ23 Vector Register
30	Write Bank A ICW1, OCW2 or OCW3 Read Bank A Poll, Interrupt Request or In-Service Status Register
31	Write Bank A ICW2, ICW3, ICW4 or OCW1 Read Bank A Interrupt Mask Register
32	Read Bank A ICW2
38	Read/Write IRQ0 Vector Register
39	Read/Write IRQ1 Vector Register
3A	Read/Write IRQ1.5 Vector Register
3B	Read/Write IRQ3 Vector Register
3C	Read/Write IRQ4 Vector Register
3D	Reserved
3E	Reserved
3F	Read/Write IRQ7 Vector Register

Port Address (HEX)	Description
PROGRAMMABLE INTERVAL TIMER	
40	Read/Write Counter 0 Register
41	Read/Write Counter 1 Register
42	Read/Write Counter 2 Register
43	Write Control Word Register I—Counter 0, 1, 2
44	Read/Write Counter 3 Register
47	Write Word Register II—Counter 3
CPU RESET	
64	Write CPU Reset Register (Data—1111XXX0H)
WAIT STATE GENERATOR	
72	Read/Write Wait State Register 0
73	Read/Write Wait State Register 1
74	Read/Write Wait State Register 2
75	Read/Write Refresh Wait State Register
DRAM REFRESH CONTROLLER	
1C	Read/Write Refresh Control Register
INTERNAL CONTROL AND DIAGNOSTIC PORTS	
61	Write Internal Control Port
80	Read/Write Internal Diagnostic Port 0
88	Read/Write Internal Diagnostic Port 1
RELOCATION REGISTER	
7F	Read/Write Relocation Register
INTEL RESERVED PORTS	
10	Reserved
12	Reserved
14	Reserved
16	Reserved
2A	Reserved
3D	Reserved
3E	Reserved
45	Reserved
46	Reserved
76	Reserved
77	Reserved
7D	Reserved
7E	Reserved
CC	Reserved
CD	Reserved
D0	Reserved
D2	Reserved
D4	Reserved
D6	Reserved

APPENDIX C

PROGRAMMING THE 82370 INTERRUPT CONTROLLERS

This Appendix describes two methods of programming and initializing the Interrupt Controllers of the 82370. A simple interrupt service routine is also shown which provides compatibility with the 82C59 Interrupt Controller.

The two methods of programming the 82370 Interrupt Controllers are needed to provide simple initialization procedures in different software environments. For new applications, a simple initialization and programming sequence can be used. For PC-DOS or other applications which expect 8259s, an interrupt handler for initialization traps must be provided. Once the handler is in place, all three 82370 Interrupt Controller banks can be programmed or initialized in the same manner as an 8259.

The ICW2 interrupt is generated by the 82370 when writing the ICW2 command to any of the interrupt controller banks. This interrupt is supplied to provide compatibility to existing code that expects to be programming 82C59s. The ICW2 value is stored in the ICW2 register of the associated bank, but is ignored by the controller. It is the responsibility of the ICW2 interrupt handler to read the ICW2 register and use its value to program the individual vector registers accordingly.

NEW APPLICATIONS

New applications do not generally require compatibility with previous code, or at least the code is usually easily modifiable. If the application fits this description, then the ICW2 interrupt can be ignored. This is done by initializing the interrupt controller as necessary, and before enabling CPU interrupts, removing the ICW2 interrupt request by reading the ICW2 register. Listing 1 shows the code for doing this for bank A. The same procedure can be used for the other banks.

Listing 1.
Initialization of an 82370 Interrupt Controller Bank
Without ICW2 Interrupts

```
INIT_BANK_A  proc near

    cli                      ;disable all interrupts

;initialize controller logic
    mov al,ICW1              ;begin sequence
    out 30h,al
    mov al,ICW2              ;send dummy ICW2
    out 31h,al
    mov al,ICW3              ;send ICW3 if necessary
    out 31h,al
    mov al,ICW4              ;send ICW4
    out 31h,al

    mov al,BANK_A_MASK      ;write to mask register (OCW1)
    out 31h,al

;program vector registers

    mov al,ICW2              ;IRQ0
    out 38h,al
    mov al,ICW2+1           ;IRQ1
    out 39h,al
    mov al,ICW2_VECTOR      ;IRQ1.5 (probably never used in
    out 3Ah,al                ; this system)
    mov al,ICW2+3           ;IRQ3
    out 3Bh,al
    mov al,ICW2+4           ;IRQ4
    out 3Ch,al
    mov al,ICW2+7           ;IRQ7
    out 3Fh,al

;remove ICW2 interrupt request

    in  al,31h              ;read mask register to work around
                            ; A-step errata

    in  al,32h              ;read ICW2 register to clear
                            ; interrupt request

;return to calling program

    sti                      ;re-enable interrupts
    ret

INIT_BANK_A  endp
```

OLD APPLICATIONS

In applications where 8259 compatibility is required, the ICW2 interrupt handler must be invoked whenever an interrupt controller is initialized (ICW1-ICW2-ICWn sequence). The handler's purpose is to read the ICW2 value from the ICW2 read register and write the appropriate sequence of vectors to the vector registers. Listing 2 shows the typical initialization sequence (this is not changed from the 8259), and the required initialization for operation of the ICW2 interrupt handler. Listing 2 shows the ICW2 interrupt handler.

Listing 2.
initialization of Bank A for ICW2 interrupts

```
cli                ;disable all interrupts

;initialize controller logic

mov al,ICW1        ;begin sequence
out 30h,al
mov al,ICW2        ;send dummy ICW2
out 31h,al

;*****
mov al,ICW3        ;send ICW3 if necessary
out 31h,al        ; note that using ICW3 for
                  ; cascading bank B is not required
                  ; and will affect the way EOIs are
                  ; required for nesting. It is
                  ; advised that ICW3 not be used.
;*****

mov al,ICW4        ;send ICW4
out ;31h,al

mov al,Bank_A_Mask ;write to mask register (OCW1=7Bh)
out 31h,al        ;don't mask off IRQ1.5 or Default
                  ; interrupt (IRQ7)

;program necessary vector registers

mov al,ICW2_VECTOR ;IRQ1.5
out 3Ah,al

mov al,IRQ7_DEFAULT_VECTOR
out 3Fh,al

;remove ICW2 interrupt request for bank A

in al,31h         ;read mask register to work around
                  ; A-step errata

in al,32h         ;read ICW2 register to clear
                  ; interrupt request

;at this point install interrupt call vector for ICW2, if
;not already done somewhere else in the code

sti                ;re-enable interrupts
```

Listing 3.
ICW2 Interrupt Service Routine

```
ICW2_INT_HANDLER      proc near

    push ax            ;save registers
    push cx
    push dx

; service bank B

    in  al,21h        ;read mask register for A-step errata

    in  al,22h        ;read ICW2
    mov cx,8          ;count vectors
    mov dx,28h        ;point to vectors

BANK_B_LOOP:

    out dx,al         ;write vector
    inc al            ;next vector
    inc dx            ;next vector I/O address
    loop BANK_B_LOOP

;service bank C

    in  al,0A1h       ;read mask register for A-step errata

    in  al,0A2h       ;read ICW2
    mov cx,8          ;count vectors
    mov dx,0A8h       ;point to vectors

BANK_C_LOOP:

    out dx,al         ;write vector
    inc al            ;next vector
    inc dx            ;next vector i/o address
    loop BANK_C_LOOP

    pop dx            ;restore registers
    pop cx
    pop ax
    iret              ;return

ICW2_INT_HANDLER      endp
```

Table 1. Interrupt Controller Registers

Bank A:

30H	write	ICW1, OCW2, OCW3
	read	Poll, IRR, ISR
31H	write	ICW2, ICW3, ICW4, OCW1
	read	IMR
32H	read	ICW2 read register
38H	read/write	IRQ0 vector
39H	read/write	IRQ1 vector
3AH	read/write	IRQ1.5 vector
3BH	read/write	IRQ3 vector
3CH	read/write	IRQ4 vector
3DH		RESERVED
3EH		RESERVED
3FH	read/write	IRQ7 vector

Bank B:

20H	write	ICW1, OCW2, OCW3
	read	Poll, IRR, ISR
21H	write	ICW2, ICW3, ICW4, OCW1
	read	IMR
22H	read	ICW2 read register
28H	read/write	IRQ8 vector
29H	read/write	IRQ9 vector
2AH		RESERVED
2BH	read/write	IRQ11 vector
2CH	read/write	IRQ12 vector
2DH	read/write	IRQ13 vector
2EH	read/write	IRQ14 vector
2FH	read/write	IRQ15 vector

Bank C:

A0H	write	ICW1, OCW2, OCW3
	read	Poll, IRR, ISR
A1H	write	ICW2, ICW3, ICW4, OCW1
	read	IMR
A2H	read	ICW2 read register
A8H	read/write	IRQ16 vector
A9H	read/write	IRQ17 vector
AAH	read/write	IRQ18 vector
ABH	read/write	IRQ19 vector
ACH	read/write	IRQ20 vector
ADH	read/write	IRQ21 vector
AEH	read/write	IRQ22 vector
AFH	read/write	IRQ23 vector

APPENDIX D SYSTEM NOTES

1. BHE# IN MASTER MODE.

In Master Mode, BHE# will be activated during DMA to/from 8-bit devices residing at even locations when the remaining byte count is greater than 1.

For example, if an 8-bit device is located at 00000000 Hex and the number of bytes to be transferred is > 1 , the first address/BHE# combination will be 00000000/0. In some systems this will cause the bus controller to perform two 8-bit accesses, the first to 00000000 Hex and the second to 00000001 Hex. However, the 82370's DMA will only read/write one byte. This may or may not cause a problem in the system depending on what is located at 00000001 Hex.

Solution:

There are two solutions if BH# active is unacceptable. Of the two, number 2 is the cleanest and most recommended.

1. If there is an 8-bit device that uses DMA located at an even address, do not use that address + 1. The limitation of this solution is that the user must have complete control over what addresses will be used in the end system.

2. Do not allow the Bus Controller to split cycles for the DMA.

2. RESET OUTPUT OF 82370:

The 80376 requires its RESET line to be active for 80 clock cycles. The 82370 generates holds the RESET line active for 62 clock cycles.

The following design example shows how the user can extend the active high of the RESET line to 80 clock cycles.

Extending the RESET Output of the 82370

This section describes a hardware solution for using the 82370's CPURST output and the software reset command to cause the 80376 to enter into a self-test.

The 80376 requires two simultaneous events in order to initiate the self-test sequence. The RESET input of the processor must be held active for at least 80 CLK2 periods and the BUSY# input must be low 8 CLK2 periods prior to and 8 CLK2 periods subsequent to RESET going inactive.

A system which does not have an 80387SX will simply have the BUSY# input to the 80376 tied low. A system which contains the 80387SX will require extra logic between the BUSY# output of the 80387SX and the BUSY# input of the 80376 in order to force self-test on reset. The extra BUSY# logic required will not be described here.

The 82370 CPURST output is intended to be retimed with faster TTL components in order to meet the RESET input setup time requirements of the 80376 and 80387SX. This requires a 74F379 (quad flip-flop with enable) or equivalent. The flip-flops required are described in TECHBIT (Ed Grochowski, April 10, 1987).

The 82370 does not meet the RESET pulse duration requirements for causing self-test of the 80376 when a software reset command is issued to the 82370. The 82370 provides a RESET pulse width of 62 CLK2 periods, the 80376 requires 80 CLK2 periods as mentioned earlier.

In order to cause the 80376 to do a self-test after a software reset, the CPURST output pulse of the 82370 must be lengthened. Figure 1 shows a circuit which will do this.

Note that the CPURST output is the OR of the 82370 RESET input and the output of the software reset command logic, and thus will have the same duration as the RESET input during power-on.

The additional circuitry required consists of an OR gate, a one-shot, a capacitor, and a resistor more than is found in a system without the 82370. The one-shot (74121) is inserted between the CPURST output of the 82370 and the input of the retiming flip-flops (74F379). The period of the one-shot should be long enough to guarantee the 80'CLK2 periods that the 80376 requires.

The OR gate (74F32) is required to guarantee that the 80376 is held in a RESET state while the 82370 is being reset. This is done to be sure that BE3# is held low when the RESET input to the 82370 goes inactive. BE3# is used during the reset to determine whether it is necessary to enter a special factory test mode. It must be low when the RESET input goes inactive, and the 80376 drives it low during reset.

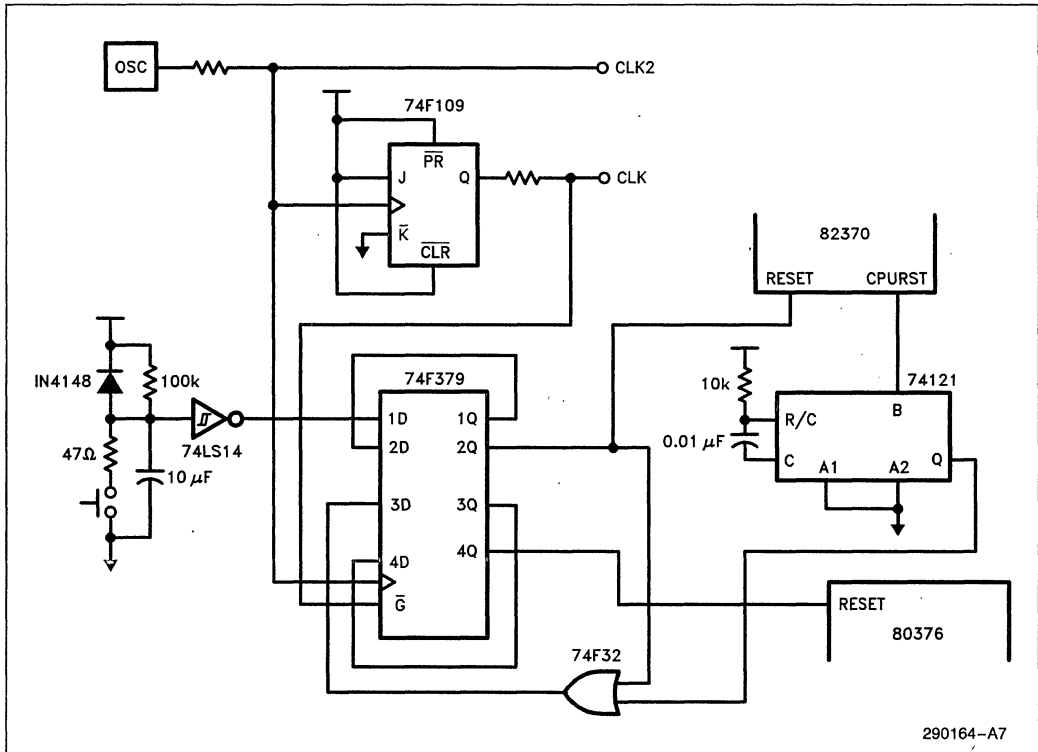


Figure D-1. Extending 82370 Reset Output

82370 TIMER UNIT NOTES

The 82370 DMA Controller with Integrated System Peripherals is functionally inconsistent with the data sheet. This document explains the behavior of the 82370 Timer Unit and outlines subsequent limitations of the timer unit. This document also provides recommended workarounds.

Overview:

There are two areas in which the 82370 timer unit exhibits non-specified behavior:

1. Mode 0 operation
2. Write Cycles to the 82370 Timer Unit

1.0 MODE 0 OPERATION

1.1 Description

For Mode 0 operation, the 82370 timer is specified as follows in the Intel 1989 Microprocessor and Peripheral Handbook Vol. I Page 4-240:

1. Writing the first byte disables counting, OUT is set LOW immediately ...

Due to mode 0 errata, this should read as follows:

1. Writing the first byte sets OUT LOW immediately. If the counter has not yet expired, writing the first byte also disables counting. However, if the counter has expired, writing the first count **does not** disable counting, although OUT still behaves correctly (set LOW immediately).

1.2 Consequences

Software errors will occur if algorithms depend on the 82370 timer unit to stop counting after writing the first byte. Thus, software that is based on the 8254 core will not function reliably on the 82370 timer unit.

Note, however, that the external signal of the timer behaves correctly.

1.3 Solution

As long as software algorithms are aware of this behavior, there should be no problems, as the external signal behaves correctly.

1.4 Long Term Plans

Currently, Intel has no plans to fix this behavior of the 82370 timer unit.

2.0 WRITE CYCLES TO THE 82370 TIMER UNIT:

This errata applies only to SLAVE WRITE cycles to the 82370 timer unit. During these cycles, the data being written into the 82370 timer unit may be corrupted if CLKIN is not inhibited during a certain "window" of the write cycle.

2.1 Description

Please refer to Figure D-2.

During write cycles to the 82370 timer unit, the 82370 translates the 80386 interface signals such as #ADS, #W/R, #M/IO, and #D/C into several internal signals that control the operation of the internal sub-blocks (e.g. Timer Unit).

The 82370 timer unit is controlled by such internal signals. These internal signals are generated and sampled with respect to two separate clock signals: CLK2 (the system clock) and CLKIN (the 82370 timer unit clock).

Since the CLKIN and CLK2 clock signals are used internally to generate control signals for the interface to the timer unit, some timing parameters must be met in order for the interface logic to function properly.

Those timing parameters are met by inhibiting the CLKIN signal for a specific window during Write Cycles to the 82370 Timer Unit.

The CLKIN signal must be inhibited using external logic, as the GATE function of the 82370 timer unit is not guaranteed to totally inhibit CLKIN.

2.2 Consequences

This CLKIN inhibits circuitry guarantees proper write cycles to the 82370 timer unit.

Without this solution, write cycles to the 82370 timer unit could place corrupted data into the timer unit registers. This, in turn, could yield inaccurate results and improper timer operation.

The proposed solution would involve a hardware modification for existing systems.

2.3 Solution

A timing waveform (Figure D-3) shows the specific window during which CLKIN must be inhibited. Please note that CLKIN must only be inhibited during the window shown in Figure D-3. This window is defined by two AC timing parameters:

$$t_a = 9 \text{ ns}$$

$$t_b = 28 \text{ ns}$$

The proposed solution provides a certain amount of system "guardband" to make sure that this window is avoided.

PAL equations for a suggested workaround are also included. Please refer to the comments in the PAL codes for stated assumptions of this particular workaround. A state diagram (Figure D-4) is provided to help clarify how this PAL is designed.

Figure D-5 shows how this PAL would fit into a system workaround. In order to show the effect of this workaround on the CLKIN signal, Figure D-6 shows how CLKIN is inhibited. Note that you must still meet the CLKIN AC timing parameters (e.g. t_{47} (min), t_{48} (min)) in order for the timer unit to function properly.

Please note that this workaround has not been tested. It is provided as a suggested solution. Actual solutions will vary from system to system.

2.4 Long Term Plans

Intel has no plans to fix this behavior in the 82370 timer unit.

```
module Timer_82370_Fix
flag '-r2', '-q2', '-f1', '-t4', '-w1,3,6,5,4,16,7,12,17,18,15,14'
title '82370 Timer Unit CLKIN
      INHIBIT signal PAL Solution '
Timer_Unit_Fix device 'P16R6';
```

"This PAL inhibits the CLKIN signal (that comes from an oscillator)
"during Slave Writes to the 82370 Timer unit.
"

"ASSUMPTION: This PAL assumes that an external system address
"decoder provides a signal to indicate that an 82370
"Timer Unit access is taking place. This input
"signal is called TMR in this PAL. This PAL also
"assumes that this TMR signal occurs during a
"specific T-State. Please see Figure 2 of this
"document to see when this signal is expected to
"be active by this PAL.
"

"NOTE: This PAL does not support pipelined 82370 SLAVE
"cycles.
"

"(c) Intel Corporation 1989. This PAL is provided as a proposed
"method of solving a certain 82370 Timer Unit problem. This PAL
"has not been tested or validated. Please validate this solution
"for your system and application.
"

"Input Pins"

CLK2	pin	1;	"System Clock
RESET	pin	2;	"Microprocessor RESET signal
TMR	pin	3;	"Input from Address Decoder, indicating "an access to the timer unit of the "82370.
!RDY	pin	4;	"End of Cycle indicator
!ADS	pin	5;	"Address and control strobe
CLK	pin	6;	"PHI2 clock
W_R	pin	7;	"Write/Read Signal"
nc1	pin	8;	"No Connect 0"
nc3	pin	9;	"No Connect 1"
GNDa	pin	10;	"Tied to ground, documentation only
GNDb	pin	11;	"Output enable, documentation only
CLKIN_IN	pin	12;	"Input-CLKIN directly from oscillator

"Output Pins"

Q_0	pin	18;	"Internal signal only, fed back to "PAL logic"
CLKIN_OUT	pin	17;	"CLKIN signal fed to 82370 Timer Unit
INHIBIT	pin	16;	"CLKIN Inhibit signal
S0	pin	15;	"Unused State Indicator Pin
S1	pin	14;	"Unused State Indicator Pin

"Declarations"

```
Valid_ADS = ADS & CLK      ; "#ADS sampled in PH11 of 80376 T-State  
Valid_RDY = RDY & CLK      ; "#RDY sampled in PH11 of 80376 T-State  
Timer_Acc = TMR & CLK      ; "Timer Unit Access, as provided by  
                          "external Address Decoder"
```

```
State_Diagram [INHIBIT, S1, S0]
```

```
state 000:      if RESET then 000  
                else if Valid_ADS & W_R then 001  
                else 000;  
  
state 001:      if RESET then 000  
                else if Timer_Acc then 010  
                else if !Timer_Acc then 000  
                else 001;  
  
state 010:      if RESET then 000  
                else if CLK then 110  
                else 010;  
  
state 110:      if RESET then 000  
                else if CLK then 111  
                else 110;  
  
state 111:      if RESET then 000  
                else if CLK then 011  
                else 111;  
  
state 011:      if RESET then 000  
                else if Valid_RDY then 000  
                else 011;  
  
state 100:      if RESET then 000  
                else 000;  
  
state 101:      if RESET then 000  
                else 000;
```

```
EQUATIONS
```

```
Q_0 := CLKIN_IN; "Latched incoming clock. This signal is used  
                "internally to feed into the MUX-ing logic"
```

```
CLKIN_OUT := (INHIBIT & CLKIN_OUT & !RESET)  
            + (!INHIBIT & Q_0 & !RESET);
```

```
"Equation for CLKIN_OUT. This  
"feeds directly to the 82370 Timer Unit."
```

```
END
```

82370 Timer Unit CLKIN
INHIBIT signal PAL Solution
Equations for Module Timer_82370_Fix

Device Timer_Unit_Fix

—Reduced Equations:

```
!!INHIBIT := (!CLK & !!INHIBIT # CLK & S0 # RESET # IS1);
```

```
IS1 := (RESET  
# !!INHIBIT & IS1  
# CLK & !!INHIBIT & !~RDY & S0 & S1  
# ICLK & IS1  
# IS1 & ITMR  
# IS0 & IS1);
```

```
IS0 := (RESET  
# INHIBIT & IS1  
# CLK & !!INHIBIT & !~RDY & S1  
# !!INHIBIT & IS0 & S1  
# ICLK & IS0  
# !!INHIBIT & IS0 & S1  
# S0 & IS1  
# IS1 & !W_R  
# ~ADS & !S1);
```

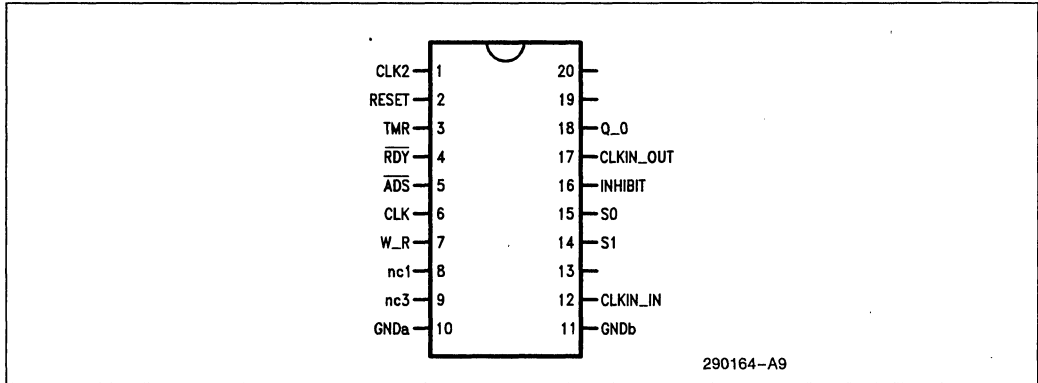
```
IQ_0 := (!CLKIN_IN);
```

```
ICLKIN_OUT := (RESET # !CLKIN_OUT & INHIBIT # !!INHIBIT & IQ_0);
```

82370 Timer Unit CLKIN
 INHIBIT signal PAL Solution
 Chip diagram for Module Timer_82370_Fix

Device Timer_Unit_Fix

P16R6



end of module Timer_82370_Fix

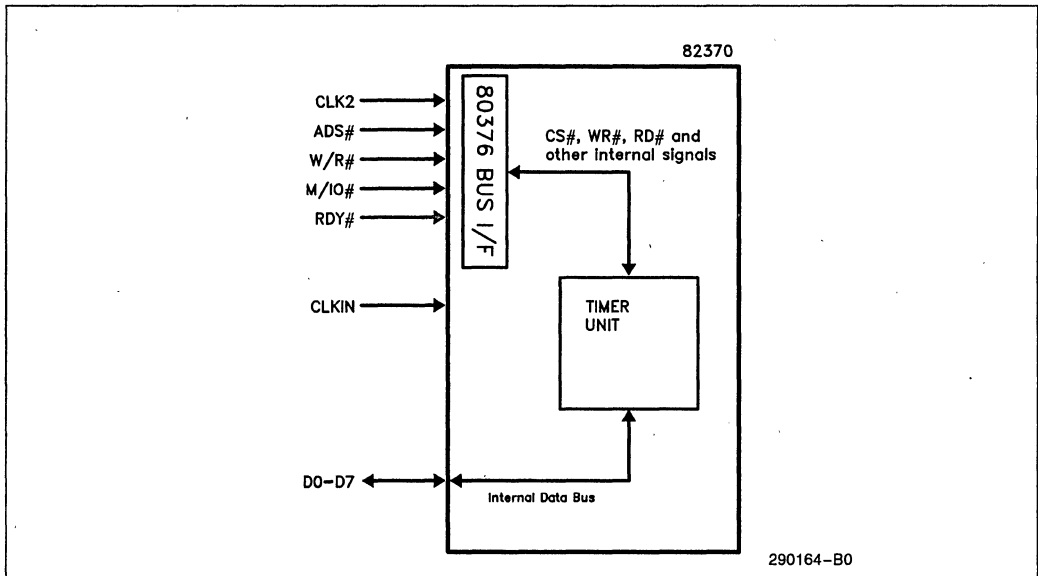


Figure D-2. Translation of 80376 Signals to Internal 82370 Timer Unit Signals

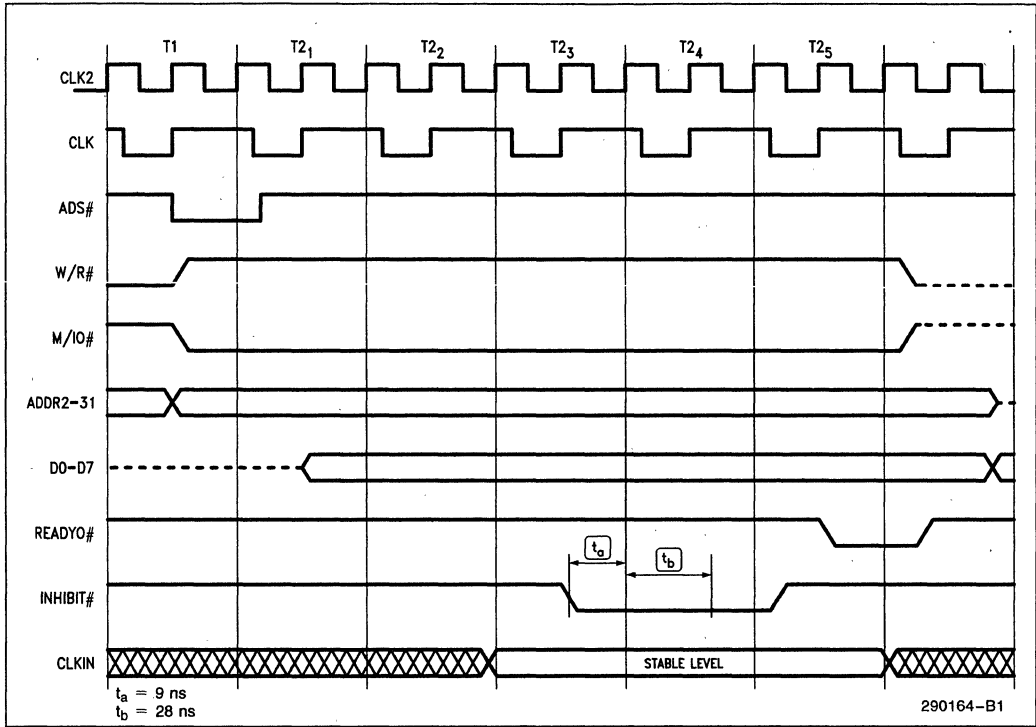


Figure D-3. 82370 Timer Unit Write Cycle

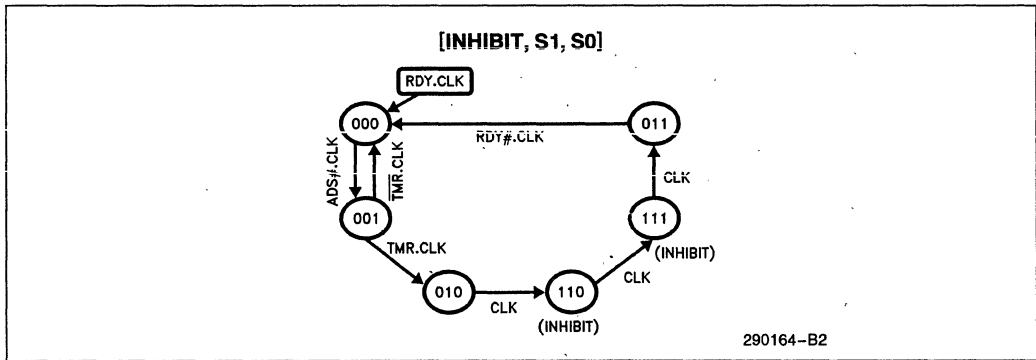


Figure D-4. State Diagram for Inhibit Signal

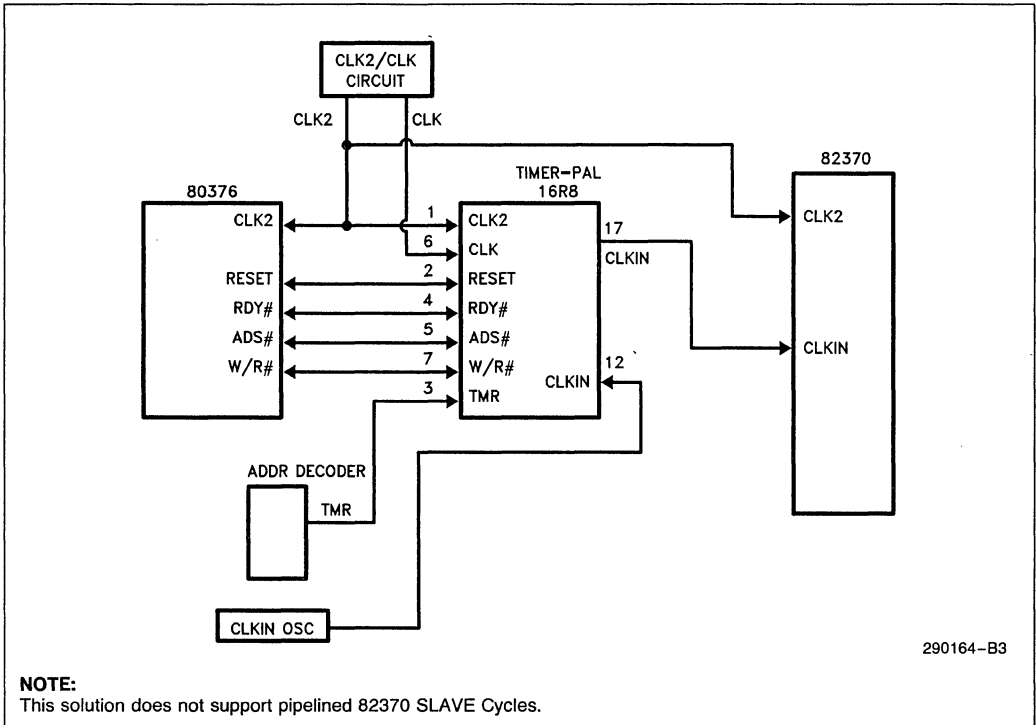


Figure D-5. System with 82370 Timer Unit "INHIBIT" Circuitry

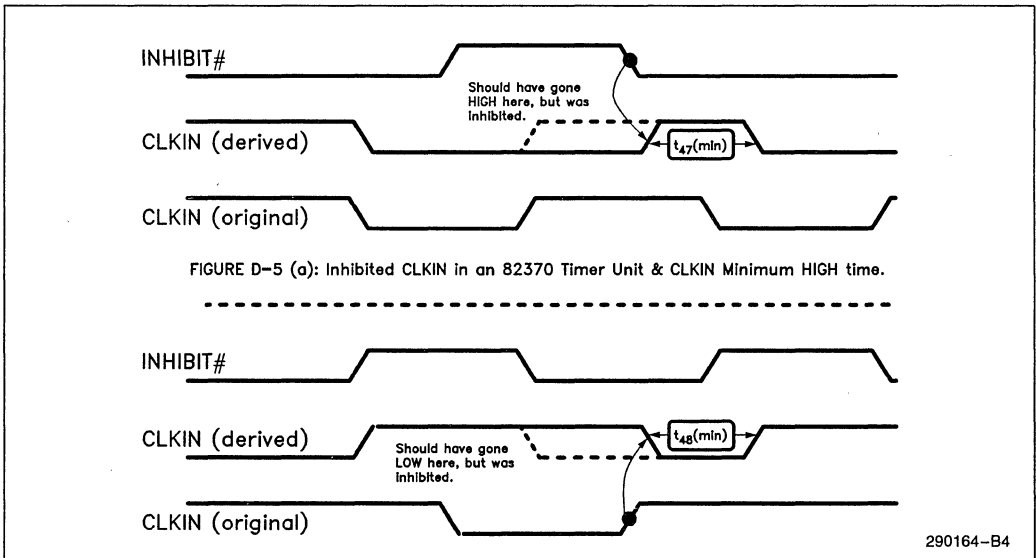


Figure D-6. Inhibited CLKIN in an 82370 Timer Unit and CLKIN Minimum LOW Time

i860™ 64-Bit Microprocessor

- **Parallel Architecture that Supports Up to Three Operations per Clock**
 - One Integer or Control Instruction per Clock
 - Up to Two Floating-Point Results per Clock
- **High Performance Design**
 - 33.3/40 MHz Clock Rates
 - 80 Peak Single Precision MFLOPs
 - 60 Peak Double Precision MFLOPs
 - 64-Bit External Data Bus
 - 64-Bit Internal Instruction Cache Bus
 - 128-Bit Internal Data Cache Bus
- **High Level of Integration on One Chip**
 - 32-Bit Integer and Control Unit
 - 32/64-Bit Pipelined Floating-Point Adder and Multiplier Units
 - 64-Bit 3-D Graphics Unit
 - Paging Unit with Translation Lookaside Buffer
 - 4 Kbyte Instruction Cache
 - 8 Kbyte Data Cache
- **Compatible with Industry Standards**
 - ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic
 - 386™/i486™ Microprocessor Data Formats and Page Table Entries
 - JEDEC 168-pin Ceramic Pin Grid Array Package (see *Packaging Outlines and Dimensions*, order # 231369)
- **Easy to Use**
 - On-Chip Debug Register
 - Assembler, Linker, Simulator, Debugger, C and FORTRAN Compilers, FORTRAN Vectorizer, Scalar and Vector Math Libraries for both OS/2* and UNIX* Environments

The Intel i860™ Microprocessor (order codes A80860-33 and A80860-40) delivers supercomputing performance in a single VLSI component. The 64-bit design of the i860 microprocessor balances integer, floating point, and graphics performance for applications such as engineering workstations, scientific computing, 3-D graphics workstations, and multiuser systems. Its parallel architecture achieves high throughput with RISC design techniques, pipelined processing units, wide data paths, large on-chip caches, million-transistor design, and fast one-micron CHMOS IV silicon technology.

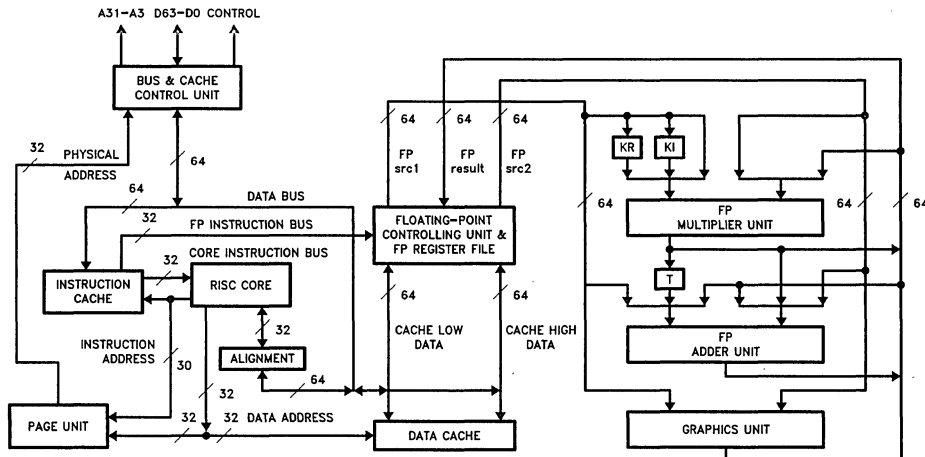


Figure 0.1. Block Diagram

240296-1

Intel, intel, 386, i486, i860, Multibus II and Parallel System Bus are trademarks of Intel Corporation.
 *UNIX is a registered trademark of AT&T. OS/2 is a trademark of International Business Machines Corporation.

1.0 FUNCTIONAL DESCRIPTION

As shown by the block diagram on the front page, the i860 microprocessor consists of 9 units:

1. Core Execution Unit
2. Floating-Point Control Unit
3. Floating-Point Adder Unit
4. Floating-Point Multiplier Unit
5. Graphics Unit
6. Paging Unit
7. Instruction Cache
8. Data Cache
9. Bus and Cache Control Unit

The core execution unit controls overall operation of the i860 microprocessor. The core unit executes load, store, integer, bit, and control-transfer operations, and fetches instructions from the floating-point unit as well. A set of 32 x 32-bit general-purpose registers are provided for the manipulation of integer data. Load and store instructions move 8-, 16-, and 32-bit data to and from these registers. Its full set of integer, logical, and control-transfer instructions give the core unit the ability to execute complete systems software and applications programs. A trap mechanism provides rapid response to exceptions and external interrupts. Debugging is supported by the ability to trap on data or instruction reference.

The floating-point hardware is connected to a separate set of floating-point registers, which can be accessed as 16 x 64-bit registers, or 32 x 32-bit registers. Special load and store instructions can also access these same registers as 8 x 128-bit registers. All floating-point instructions use these registers as their source and destination operands.

The floating-point control unit controls both the floating-point adder and the floating-point multiplier, issuing instructions, handling all source and result exceptions, and updating status bits in the floating-point status register. The adder and multiplier can operate in parallel, producing up to two results per clock. The floating-point data types, floating-point instructions, and exception handling all support the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985).

The floating-point adder performs addition, subtraction, comparison, and conversions on 64- and 32-bit floating-point values. An adder instruction executes in three clocks; however, in pipelined mode, a new result is generated every clock.

The floating-point multiplier performs floating-point and integer multiply and floating-point reciprocal operations on 64- and 32-bit floating-point values. A multiplier instruction executes in three to four clocks;

however, in pipelined mode, a new result can be generated every clock for single-precision and every other clock for double precision.

The graphics unit has special integer logic that supports three-dimensional drawing in a graphics frame buffer, with color intensity shading and hidden surface elimination via the Z-buffer algorithm. The graphics unit recognizes the pixel as an 8-, 16-, or 32-bit data type. It can compute individual red, blue, and green color intensity values within a pixel; but it does so with parallel operations that take advantage of the 64-bit internal word size and 64-bit external bus. The graphics features of the i860 microprocessor assume that the surface of a solid object is drawn with polygon patches whose shapes approximate the original object. The color intensities of the vertices of the polygon and their distances from the viewer are known, but the distances and intensities of the other points must be calculated by interpolation. The graphics instructions of the i860 microprocessor directly aid such interpolation.

The paging unit implements protected, paged, virtual memory via a 64-entry, four-way set-associative memory called the TLB (Translation Lookaside Buffer). The paging unit uses the TLB to perform the translation of logical address to physical address, and to check for access violations. The access protection scheme employs two levels of privilege: user and supervisor.

The instruction cache is a two-way set-associative memory of four Kbytes, with 32-byte blocks. It transfers up to 64 bits per clock (320 Mbyte/sec at 40 MHz).

The data cache is a two-way set-associative memory of eight Kbytes, with 32-byte blocks. It transfers up to 128 bits per clock (640 Mbyte/sec at 40 MHz). The i860 microprocessor normally uses writeback caching, i.e. memory writes update the cache (if applicable) without necessarily updating memory immediately; however, caching can be inhibited by software where necessary.

The bus and cache control unit performs data and instruction accesses for the core unit. It receives cycle requests and specifications from the core unit, performs the data-cache or instruction-cache miss processing, controls TLB translation, and provides the interface to the external bus. Its pipelined structure supports up to three outstanding bus cycles.

2.0 PROGRAMMING INTERFACE

The programmer-visible aspects of the architecture of the i860 microprocessor include data types, registers, instructions, and traps.

2.1 Data Types

The i860 microprocessor provides operations for integer and floating-point data. Integer operations are performed on 32-bit operands with some support also for 64-bit operands. Load and store instructions can reference 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit operands. Floating-point operations are performed on IEEE-standard 32- and 64-bit formats. Graphics oriented instructions operate on arrays of 8-, 16-, or 32-bit pixels.

2.1.1 INTEGER

An integer is a 32-bit signed value in standard two's complement form. A 32-bit integer can represent a value in the range $-2,147,483,648$ (-2^{31}) to $2,147,483,647$ ($+2^{31} - 1$). Arithmetic operations on 8- and 16-bit integers can be performed by sign-extending the 8- or 16-bit values to 32 bits, then using the 32-bit operations.

There are also add and subtract instructions that operate on 64-bit long integers.

Load and store instructions may also reference (in addition to the 32- and 64-bit formats previously mentioned) 8- and 16-bit items in memory. When an 8- or 16-bit item is loaded into a register, it is converted to an integer by sign-extending the value to 32 bits. When an 8- or 16-bit item is stored from a register, the corresponding number of low-order bits of the register are used.

2.1.2 ORDINAL

Arithmetic operations are available for 32-bit ordinals. An ordinal is an unsigned integer. An ordinal can represent values in the range 0 to $4,294,967,295$ ($+2^{32} - 1$).

Also, there are add and subtract instructions that operate on 64-bit ordinals.

2.1.3 SINGLE- AND DOUBLE-PRECISION REAL

Figure 2.1 shows the real number formats. A single-precision real (also called "single real") data type is a 32-bit binary floating-point number. Bit 31 is the sign bit; bits 30..23 are the exponent; and bits 22..0 are the fraction. In accordance with ANSI/IEEE standard 754, the value of a single-precision real is defined as follows:

1. If $e = 0$ and $f \neq 0$ or $e = 255$ then generate a floating-point source-exception trap when encountered in a floating-point operation.
2. If $0 < e < 255$, then the value is $(-1)^s \times 1.f \times 2^{e-127}$.
3. If $e = 0$ and $f = 0$, then the value is signed zero.

A double-precision real (also called "double real") data type is a 64-bit binary floating-point number. Bit 63 is the sign bit; bits 62..52 are the exponent; and bits 51..0 are the fraction. In accordance with ANSI/IEEE standard 754, the value of a double-precision real is defined as follows:

1. If $e = 0$ and $f \neq 0$ or $e = 2047$, then generate a floating-point source-exception trap when encountered in a floating-point operation.
2. If $0 < e < 2047$, then the value is $(-1)^s \times 1.f \times 2^{e-1023}$.

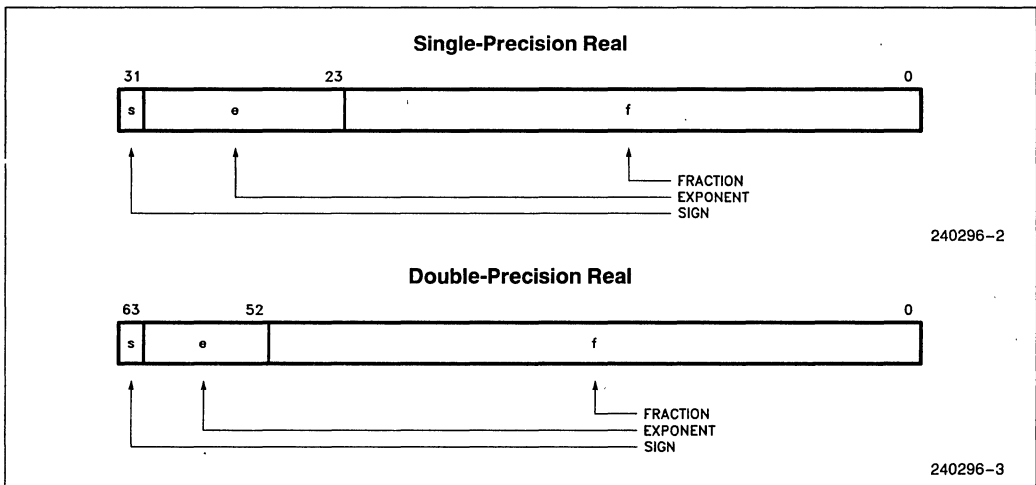


Figure 2.1. Real Number Formats

3. If $e = 0$ and $f = 0$, then the value is signed zero.

The special values infinity, NaN (“Not a Number”), indefinite, and denormal generate a trap when encountered. The trap handler implements IEEE-standard results.

A double real value occupies an even/odd pair of floating-point registers. Bits 31..0 are stored in the even-numbered floating-point register; bits 63..32 are stored in the next higher odd-numbered floating-point register.

2.1.4 PIXEL

A pixel may be 8, 16, or 32 bits long depending on color and intensity resolution requirements. Regardless of the pixel size, the i860 microprocessor always operates on 64 bits worth of pixels at a time. The pixel data type is used by two kinds of instructions:

- The selective pixel-store instruction that helps implement hidden surface elimination.
- The pixel add instruction that helps implement 3-D color intensity shading.

To perform color intensity shading efficiently in a variety of applications, the i860 microprocessor defines three pixel formats according to Table 2.1.

Figure 2.2 illustrates one way of assigning meaning to the fields of pixels. These assignments are for illustration purposes only. The i860 microprocessor defines only the field sizes, not the specific use of each field. Other ways of using the fields of pixels are possible.

Table 2.1. Pixel Formats

Pixel Size (in bits)	Bits of Color 1 Intensity	Bits of Color 2 Intensity	Bits of Color 3 Intensity	Bits of Other Attribute (Texture)
8	N (≤ 8) bits of intensity*			8 - N
16	6	6	4	
32	8	8	8	8

The intensity attribute fields may be assigned to colors in any order convenient to the application.

*With 8-bit pixels, up to 8 bits can be used for intensity; the remaining bits can be used for any other attribute, such as color. The intensity bits must be the low-order bits of the pixel.

2.2 Register Set

As Figure 2.3 shows, the i860 microprocessor has the following registers:

- An integer register file
- A floating-point register file
- Six control registers (**psr**, **epsr**, **db**, **dirbase**, **fir**, and **fsr**)
- Four special-purpose registers (KR, KI, T, and MERGE)

The control registers are accessible only by load and store control-register instructions; the integer and floating-point registers are accessed by arithmetic operations and load and store instructions. The special-purpose registers KR, KI, T, and MERGE are used by a few specific instructions.

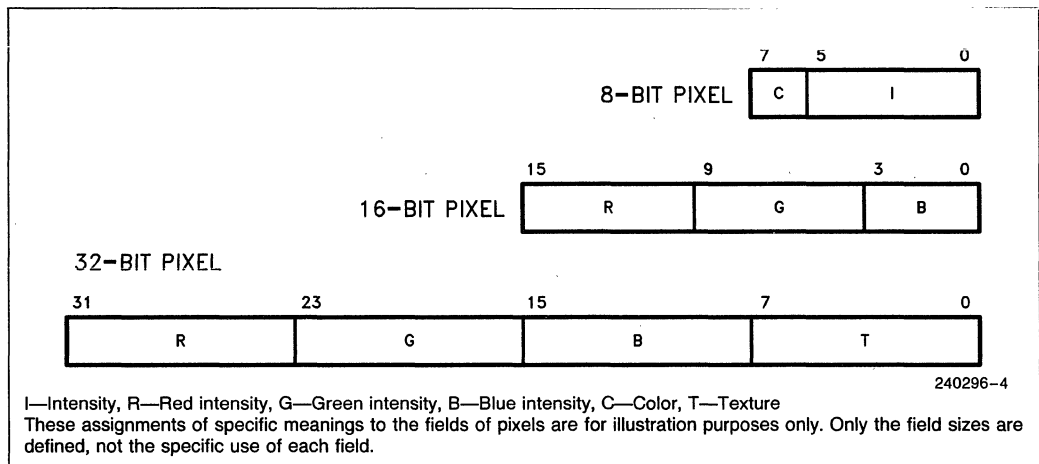


Figure 2.2. Pixel Format Example

2.2.1 INTEGER REGISTER FILE

There are 32 integer registers, each 32 bits wide, referred to as **r0** through **r31**, which are used for address computation and scalar integer computations. Register **r0** always returns zero when read, independently of what is stored in it.

2.2.2 FLOATING-POINT REGISTER FILE

There are 32 floating-point registers, each 32-bits wide, referred to as **f0** through **f31**, which are used for floating-point computations. Registers **f0** and **f1** always return zero when read, independently of what is stored in them. The floating-point registers are also used by a set of graphics operations, primarily for 3D graphics computations.

When accessing 64-bit floating-point or integer values, the i860 microprocessor uses an even/odd pair of registers. When accessing 128-bit values, it uses an aligned set of four registers (**f0**, **f4**, **f8**, . . . , **f28**). The instruction must designate the lowest register number of the set of registers containing 64- or 128-bit values. Misaligned register numbers produce undefined results. The register with the lowest number contains the least significant part of the value. For 128-bit values, the register pair with the lower numbers contain the least significant 64 bits while the register pair with the higher numbers contain the most significant 64 bits.

The 128-bit load and store instructions, along with the 128-bit data path between the floating-point registers and the data cache help to sustain the extraordinarily high rate of computation.

2.2.3 PROCESSOR STATUS REGISTER

The processor status register (**psr**) contains miscellaneous state information for the current process. Figure 2.4 shows the format of the **psr**.

- **BR** (Break Read) and **BW** (Break Write) enable a data access trap when the operand address matches the address in the **db** register and a read or write (respectively) occurs.
- Various instructions set **CC** (Condition Code) according to tests they perform. The branch-on-condition-code instructions test its value. The **bla** instruction sets and tests **LCC** (Loop Condition Code).
- **IM** (Interrupt Mode) enables external interrupts if set; disables interrupts if clear.
- **U** (User Mode) is set when the i860 microprocessor is executing in user mode; it is clear when the i860 microprocessor is executing in supervisor mode. In user mode, writes to some control registers are inhibited. This bit also controls the memory protection mechanism. See section 2.4.4.3 for a description of memory protection in user and supervisor modes.

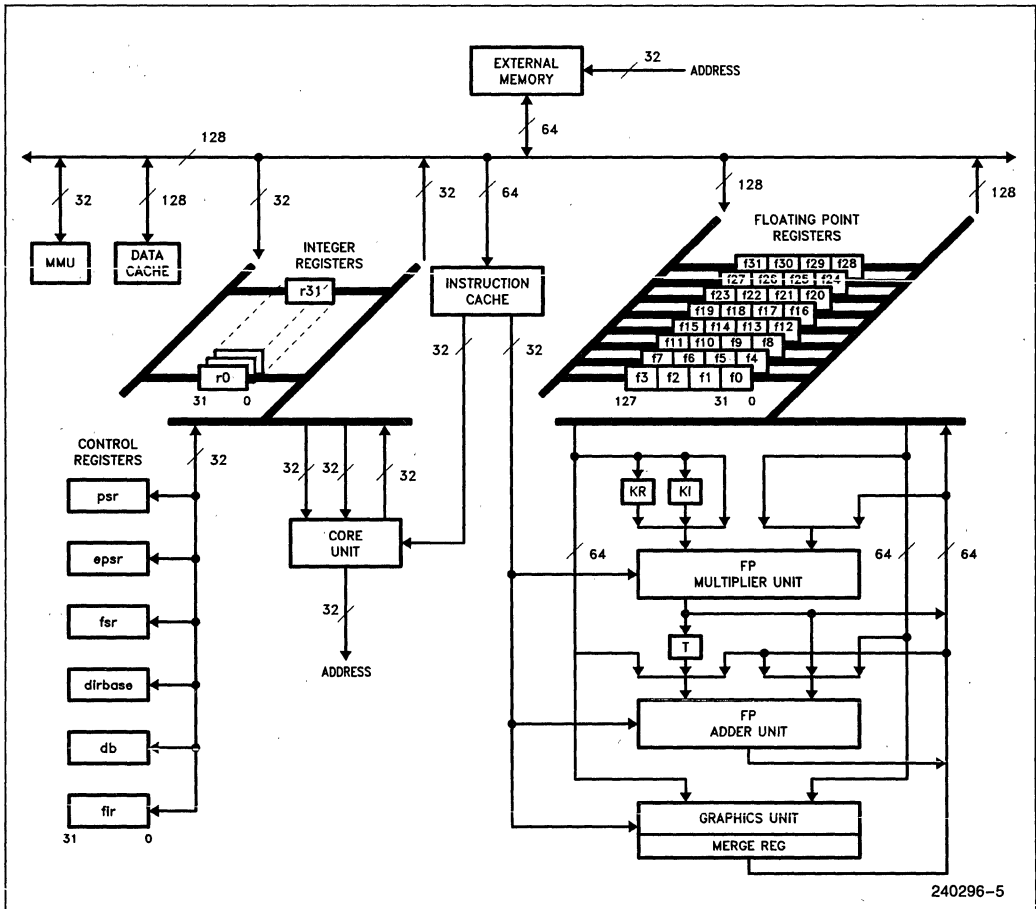


Figure 2.3. Registers and Data Paths

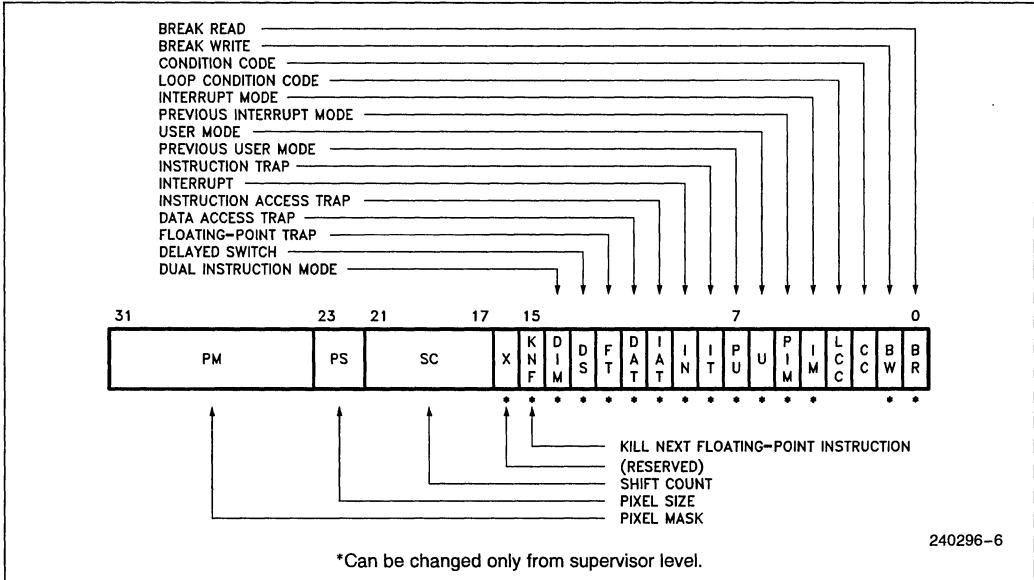


Figure 2.4 Processor Status Register

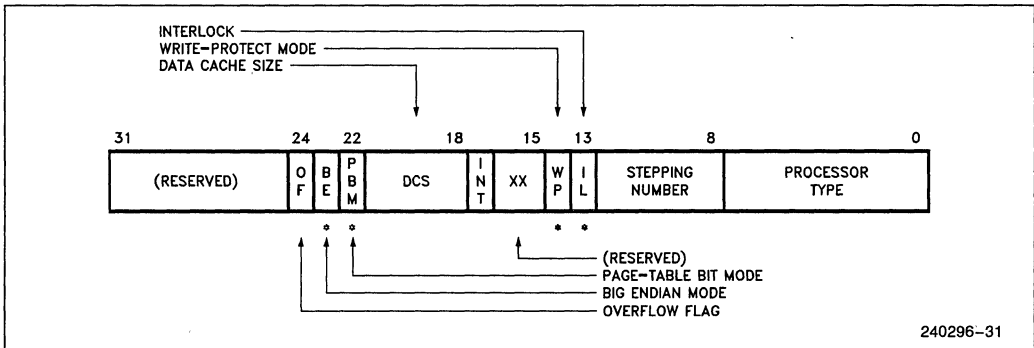


Figure 2.5 Extended Processor Status Register

- PIM (Previous Interrupt Mode) and PU (Previous User Mode) save the corresponding status bits (IM and U) on a trap, because those status bits are changed when a trap occurs. They are restored into their corresponding status bits when returning from a trap handler with a branch indirect instruction when a trap flag is set in the **psr**.
- FT (Floating-Point Trap), DAT (Data Access Trap), IAT (Instruction Access Trap), IN (Interrupt), and IT (Instruction Trap) are trap flags. They are set when the corresponding trap condition occurs. The trap handler examines these bits to determine which condition or conditions have caused the trap.
- DS (Delayed Switch) is set if a trap occurs during the instruction before dual-instruction mode is entered or exited. If DS is set and DIM (Dual Instruction Mode) is clear, the i860 microprocessor switches to dual-instruction mode one instruction after returning from the trap handler. If DS and DIM are both set, the i860 microprocessor switches to single-instruction mode one instruction after returning from the trap handler.
- When a trap occurs, the i860 microprocessor sets DIM if it is executing in dual-instruction mode; it clears DIM if it is executing in single-instruction mode. If DIM is set after returning from a trap handler, the i860 microprocessor resumes execution in dual-instruction mode.

- When KNF (Kill Next Floating-Point Instruction) is set, the next floating-point instruction is suppressed (except that its dual-instruction mode bit is interpreted). A trap handler sets KNF if the trapped floating-point instruction should not be reexecuted.
- SC (Shift Count) stores the shift count used by the last right-shift instruction. It controls the number of shifts executed by the double-shift instruction.
- PS (Pixel Size) and PM (Pixel Mask) are used by the pixel-store instruction and by the graphics instructions. The values of PS control pixel size as defined by Table 2.2. The bits in PM correspond to pixels to be updated by the pixel-store instruction **pst.d**. The low-order bit of PM corresponds to the low-order pixel of the 64-bit source operand of **pst.d**. The number of low-order bits of PM that are actually used is the number of pixels that fit into 64-bits, which depends upon PS. If a bit of PM is set, then **pst.d** stores the corresponding pixel. Refer also to the **pst.d** instruction in section 8.

Table 2.2. Values of PS

Value	Pixel Size in bits	Pixel Size in bytes
00	8	1
01	16	2
10	32	4
11	(undefined)	(undefined)

2.2.4 EXTENDED PROCESSOR STATUS REGISTER

The extended processor status register (**epsr**) contains additional state information for the current process beyond that stored in the **psr**. Figure 2.5 shows the format of the **epsr**.

- The processor type is one for the i860 microprocessor.
- The stepping number has a unique value that distinguishes among different revisions of the processor.
- IL (Interlock) is set if a trap occurs after a **lock** instruction but before the load or store following the subsequent **unlock** instruction. IL indicates to the trap handler that a locked sequence has been interrupted. This does not apply if a **st.c dirbase** was used to set the BL bit.
- WP (write protect) controls the semantics of the W bit of page table entries. A clear W bit in either the directory or the page table entry causes writes to be trapped. When WP is clear, writes are trapped in user mode, but not in supervisor mode. When WP is set, writes are trapped in both user and supervisor modes.
- INT (Interrupt) is the value of the INT input pin.
- DCS (Data Cache Size) is a read-only field that tells the size of the on-chip data cache. The number of bytes actually available is 2^{12+DCS} ; therefore, a value of zero indicates 4 Kbytes, one indicates 8 Kbytes, etc.

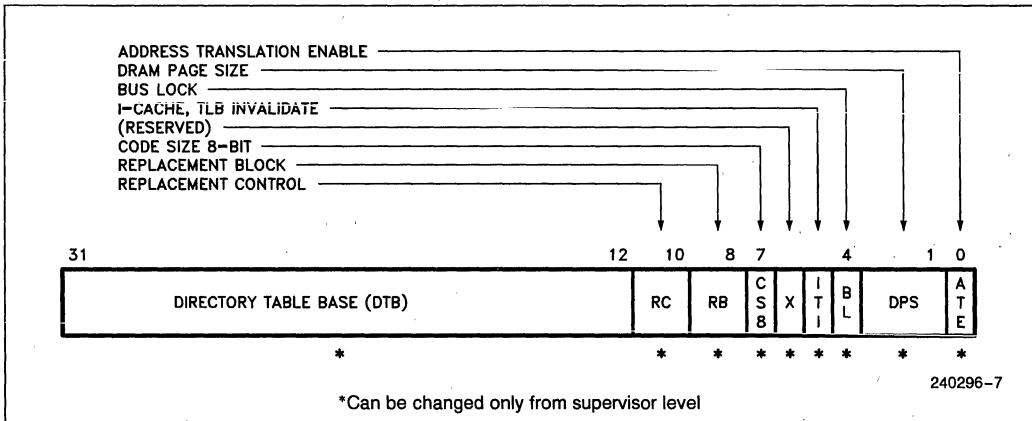


Figure 2.6. Directory Base Register

- PBM (Page-Table Bit Mode) determines which bit of page-table entries is output on the PTB pin. When PBM is clear, the PTB signal reflects bit CD of the page-table entry used for the current cycle. When PBM is set, the PTB signal reflects bit WT of the page-table entry used for the current cycle.
- BE (Big Endian) controls the ordering of bytes within a data item in memory. Normally (i.e. when BE is clear) the i860 microprocessor operates in little endian mode, in which the addressed byte is the low-order byte. When BE is set (big endian mode), the low-order three bits of all load and store addresses are complemented, then masked to the appropriate boundary for alignment. This causes the addressed byte to be the most significant byte. Section 2.3 discusses little and big endian addressing.
- OF (Overflow Flag) is set by **adds**, **addu**, **subs**, and **subu** when integer overflow occurs. For **adds** and **subs**, OF is set if the carry from bit 31 is different than the carry from bit 30. For **addu**, OF is set if there is a carry from bit 31. For **subu**, OF is set if there is no carry from bit 31. Under all other conditions, it is cleared by these instructions. OF controls the function of the **intovr** instruction.
- DPS (DRAM Page Size) controls how many bits to ignore when comparing the current bus-cycle address with the previous bus-cycle address to generate the NENE# signal. This feature allows for higher speeds when using static column or page-mode DRAMs and consecutive reads and writes access the row. The comparison ignores the low-order 12 + DPS bits. A value of zero is appropriate for one bank of $256K \times n$ RAMs, 1 for $1M \times n$ RAMs, etc.
- When BL (Bus Lock) is set, external bus accesses are locked. The LOCK# signal is asserted the next bus cycle whose internal bus request is generated after BL is set. It remains set on every subsequent bus cycle as long as BL remains set. The LOCK# signal is deasserted on the next internal bus request generated after BL is cleared. Traps immediately clear BL. The **lock** and **unlock** instructions control the BL bit.
- ITI (I-Cache, TLB Invalidate), when set in the value that is loaded into **dirbase**, causes the instruction cache and address-translation cache (TLB) to be flushed. The ITI bit does not remain set in **dirbase**. ITI always appears as zero when reading **dirbase**. Section 2.5 discusses flushing the data cache before invalidating the TLB.
- When CS8 (Code Size 8-Bit) is set, instruction cache misses are processed as 8-bit bus cycles. When this bit is clear, instruction cache misses are processed as 64-bit bus cycles. This bit can not be set by software; hardware sets this bit at initialization time. It can be cleared by software (one time only) to allow the system to execute out of 64-bit memory after bootstrapping from 8-bit EPROM. A nondelayed branch to code in 64-bit memory should directly follow the **st.c** (store control register) instruction that clears CS8, in order to make the transition from 8-bit to 64-bit memory occur at the correct time. The branch must be aligned on a 64-bit boundary.

2.2.5 DATA BREAKPOINT REGISTER

The data breakpoint register (**db**) is used to generate a trap when the i860 microprocessor makes a data-operand access to the address stored in this register. The trap is enabled by BR and BW in **psr**. The **db** register can only be changed from supervisor level. When comparing, a number of low order bits of the address are ignored, depending on the size of the operand. For example, a 16-bit access ignores the low-order bit of the address when comparing to **db**; a 32-bit access ignores the low-order two bits. This ensures that any access that overlaps the address contained in the register will generate a trap. The DAT occurs before the data is accessed and prevents the load or store from completing.

2.2.6 DIRECTORY BASE REGISTER

The directory base register **dirbase** (shown in Figure 2.6) controls address translation, caching, and bus options. The **dirbase** register can only be changed from supervisor level. The BL bit is changed from user level with the **lock** and **unlock** instructions.

- ATE (Address Translation Enable), when set, enables the virtual-address translation algorithm. The data cache must be flushed before changing the ATE bit.
- RB (Replacement Block) identifies the cache block to be replaced by cache replacement algorithms. The high-order bit of RB is ignored by the instruction and data caches. RB conditions the cache flush instruction **flush**, which is discussed in Section 8. Table 2.3 explains the values of RB.
- RC (Replacement Control) controls cache replacement algorithms. Table 2.4 explains the significance of the values of RC.
- DTB (Directory Table Base) contains the high-order 20 bits of the physical address of the page directory when address translation is enabled (i.e. ATE = 1). The low-order 12 bits of the address are zeros.

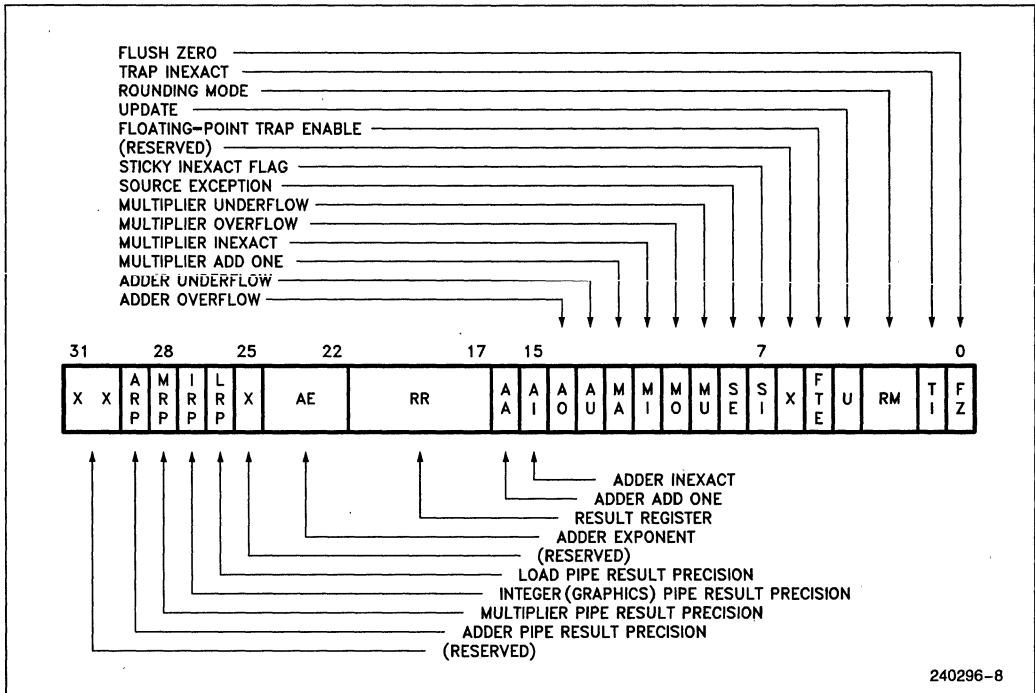


Figure 2.7. Floating-Point Status Register

Table 2.3. Values of RB

Value	Replace TLB Block	Replace Instruction and Data Cache Block
0 0	0	0
0 1	1	1
1 0	2	0
1 1	3	i

Table 2.4. Values of RC

Value	Meaning
00	Selects the normal replacement algorithm where any block in the set may be replaced on cache misses in all caches.
01	Instruction, data, and TLB cache misses replace the block selected by RB. The instruction and data caches ignore the high-order bit of RB. This mode is used for instruction cache and TLB testing.
10	Data cache misses replace the block selected by the low-order bit of RB.
11	Disables data cache replacement.

2.2.7 FAULT INSTRUCTION REGISTER

When a trap occurs, this register contains the address of the trapping instruction (not necessarily the instruction that created the conditions that required the trap). The fir is a read only register. The address of the *id.c* instruction used to read the fir is returned in *rdest* when reading the fir at any time other than the first *ld.c* fir after a trap.

2.2.8 FLOATING-POINT STATUS REGISTER

The floating-point status register (*fsr*) contains the floating-point trap and rounding-mode status for the current process. Figure 2.7 shows its format. The *fsr* is writable in user level.

- If FZ (Flush Zero) is clear and underflow occurs, a result-exception trap is generated. When FZ is set and underflow occurs, the result is set to zero, and no trap due to underflow occurs.
- If TI (Trap Inexact) is clear, inexact results do not cause a trap. If TI is set, inexact results cause a trap. The sticky inexact flag (SI) is set whenever an inexact result is produced, regardless of the setting of TI.
- RM (Rounding Mode) specifies one of the four rounding modes defined by the IEEE standard. Given a true result *b* that cannot be represented

Table 2.5. Values of RM

Value	Rounding Mode	Rounding Action
00	Round to nearest or even	Closer to b of a or c ; if equally close, select even number (the one whose least significant bit is zero).
01	Round down (toward $-\infty$)	a
10	Round up (toward $+\infty$)	c
11	Chop (toward zero)	Smaller in magnitude of a or c .

by the target data type, the i860 microprocessor determines the two representable numbers a and c that most closely bracket b in value ($a < b < c$). The i860 microprocessor then rounds (changes) b to a or c according to the mode selected by RM as defined in Table 2.5. Rounding introduces an error in the result that is less than one least-significant bit.

- The U-bit (Update Bit), if set in the value that is loaded into **fsr** by a **st.c** instruction, enables updating of the result-status bits (AE, AA, AI, AO, AU, MA, MI, MO, and MU) in the first-stage of the floating-point adder and multiplier pipelines. If this bit is clear, the result-status bits are unaffected by a **st.c** instruction; **st.c** ignores the corresponding bits in the value that is being loaded. A **st.c** always updates **fsr** bits 21..17 and 8..0 directly. The U-bit does not remain set; it always appears as zero when read.
- The FTE (Floating-Point Trap Enable) bit, if clear, disables all floating-point traps (invalid input operand, overflow, underflow, and inexact result).
- SI (Sticky Inexact) is set when the last stage result of either the multiplier or adder is inexact (i.e. when either AI or MI is set). SI is “sticky” in the sense that it remains set until reset by software. AI and MI, on the other hand, can be changed by the subsequent floating-point instruction.
- SE (Source Exception) is set when one of the source operands of a floating-point operation is invalid; it is cleared when all the input operands are valid. Invalid input operands include denormals, infinities, and all NaNs (both quiet and signaling).
- When read from the **fsr**, the result-status bits MA, MI, MO, and MU (Multiplier Add-One, Inexact, Overflow, and Underflow, respectively) describe the last stage result of the multiplier.

When read from the **fsr**, the result-status bits AA, AI, AO, AU, and AE (Adder Add-One, Inexact, Overflow, Underflow, and Exponent, respectively) describe the last stage result of the adder. The high-order three bits of the 11-bit exponent of the adder result are stored in the AE field.

The Adder Add One and Multiplier Add One bits indicate that the absolute value of the result frac-

tion grew by one least-significant bit due to rounding. AA and MA are not influenced by the sign of the result.

After a floating-point operation in a given unit (adder or multiplier), the result-status bits of that unit are undefined until the point at which result exceptions are reported.

When written to the **fsr** with the U-bit set, the result-status bits are placed into the first stage of the adder and multiplier pipelines. When the processor executes pipelined operations, it propagates the result-status bits of a particular unit (multiplier or adder) one stage for each pipelined floating-point operation for that unit. When they reach the last stage, they replace the normal result-status bits in the **fsr**. When the U-bit is not set, result-status bits in the word being written to the **fsr** are ignored.

In a floating-point dual-operation instruction (e.g. add-and-multiply or subtract-and-multiply), both the multiplier and the adder may set exception bits. The result-status bits for a particular unit remain set until the next operation that uses that unit.

- RR (Result Register) specifies which floating-point register (**f0–f31**) was the destination register when a result-exception trap occurs due to a scalar operation.
- LRP (Load Pipe Result Precision), IRP (Integer (Graphics) Pipe Result Precision), MRP (Multiplier Pipe Result Precision), and ARP (Adder Pipe Result Precision) aid in restoring pipeline state after a trap or process switch. Each defines the precision of the last stage result in the corresponding pipeline. One of these bits is set when the result in the last stage of the corresponding pipeline is double precision; it is cleared if the result is single precision. These bits cannot be changed by software.

2.2.9 KR, KI, T, AND MERGE REGISTERS

The KR, KI, and T registers are special-purpose registers used by the dual-operation floating-point instructions **pfam**, **pfam**, **pfsm**, and **pfmsm**,

which initiate both an adder (A-unit) operation and a multiplier (M-unit) operation. The KR, KI, and T registers can store values from one dual-operation instruction and supply them as inputs to subsequent dual-operation instructions. (Refer to Figure 2.14.)

The MERGE register is used only by the graphics instructions. The purpose of the MERGE register is to accumulate (or merge) the results of multiple-addition operations that use as operands the color-intensity values from pixels or distance values from a Z-buffer. The accumulated results can then be stored in one 64-bit operation.

Two multiple-addition instructions and an OR instruction use the MERGE register. The addition instructions are designed to add interpolation values to each color-intensity field in an array of pixels or to each distance value in a Z-buffer.

Refer to the instruction descriptions in section 8 for more information about these registers.

2.3 Addressing

Memory is addressed in byte units with a paged virtual-address space of 2^{32} bytes. Data and instructions can be located anywhere in this address space. Address arithmetic is performed using 32-bit input values and produces 32-bit results. The low-order 32 bits of the result are used in case of overflow.

Normally, multibyte data values are stored in memory in little endian format, i.e., with the least significant byte at the lowest memory address. As an option, the ordering can be dynamically selected by software in supervisor mode. The i860 microprocessor also offers big endian mode, in which the most significant byte of a data item is at the lowest address. Figure 2.8 shows the difference between the two storage modes. Big endian and little endian data areas should not be mixed within a 64-bit data word. Illustrations of data structures in this data sheet show data stored in little endian mode, i.e., the low-order byte is at the lowest memory address.

Code accesses are always done with little endian addressing. This implies that code will appear differently than documented here when accessed as big endian data. Intel recommends that disassemblers running in a big endian system, convert instructions which have been read as data back to little endian form and present them in the format documented here.

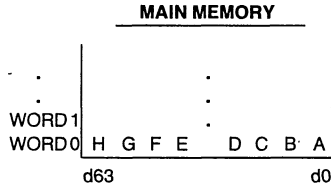
Alignment requirements are as follows (any violation results in a data-access trap):

- 128-bit values are aligned on i6-byte boundaries when referenced in memory (i.e. the four least significant address bits must be zero).
- 64-bit values are aligned on 8-byte boundaries when referenced in memory (i.e. the three least significant address bits must be zero).
- 32-bit values are aligned on 4-byte boundaries when referenced in memory (i.e. the two least significant address bits must be zero).
- 16-bit values are aligned on 2-byte boundaries when referenced in memory (i.e. the least significant address bit must be zero).

2.4 Virtual Addressing

When address translation is enabled, the i860 microprocessor maps instruction and data virtual addresses into physical addresses before referencing memory. This address transformation is compatible with that of the 386 microprocessor and implements the basic features needed for page-oriented virtual-memory systems and page-level protection.

The address translation is optional. Address translation is in effect only when the ATE bit of `dirbase` is set. This bit is typically set by the operating system during software initialization. The ATE bit must be set if the operating system is to implement page-oriented protection or page-oriented virtual memory.



	LITTLE ENDIAN			BIG ENDIAN		
	Byte Enables (BE#)	DATA BUS d63 d0	r16 d31 d0	Byte Enables (BE#)	DATA BUS d63 d0	r16 d31 d0
ld.b 0(r0), r16	0			7		
ld.b 1(r0), r16	1			6		
ld.b 2(r0), r16	2			5		
ld.b 3(r0), r16	3			4		
ld.b 4(r0), r16	4			3		
ld.b 5(r0), r16	5			2		
ld.b 6(r0), r16	6			1		
ld.b 7(r0), r16	7			0		
ld.s 0(r0), r16	1:0			7:6		
ld.s 2(r0), r16	3:2			5:4		
ld.s 4(r0), r16	5:4			3:2		
ld.s 6(r0), r16	7:6			1:0		
ld.l 0(r0), r16	3:0			7:4		
ld.l 4(r0), r16	7:4			3:0		

NOTE:

64- and 128-bit big endian accesses are treated the same as little endian accesses.

Figure 2.8 Little and Big Endian Accesses

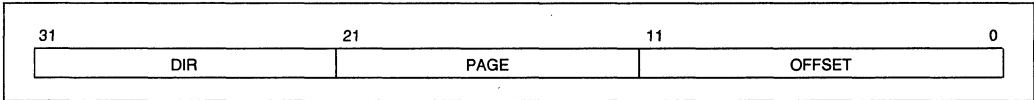


Figure 2.9. Format of a Virtual Address

Address translation is disabled when the processor is reset. It is enabled when a store to **dirbase** sets the ATE bit. It is disabled again when a store clears the ATE bit.

2.4.1 PAGE FRAME

A **page frame** is a 4-Kbyte unit of contiguous addresses of physical main memory. Page frames begin on 4-Kbyte boundaries and are fixed in size. A **page** is the collection of data that occupies a page frame when that data is present in main memory. The data may also occupy some location in secondary storage when there is not sufficient space in main memory.

2.4.2 VIRTUAL ADDRESS

A virtual address refers indirectly to a physical address by specifying a page table, a page within that

table, and an offset within that page. Figure 2.9 shows the format of a virtual address.

Figure 2.10 shows how the i860 microprocessor converts the DIR, PAGE, and OFFSET fields of a virtual address into the physical address by consulting two levels of page tables. The addressing mechanism uses the DIR field as an index into a page directory, uses the PAGE field as an index into the page table determined by the page directory, and uses the OFFSET field to address a byte within the page determined by the page table.

2.4.3 PAGE TABLES

A page table is simply an array of 32-bit page specifiers. A page table is itself a page, and therefore contains 4 Kbytes of memory or at most 1K 32-bit entries.

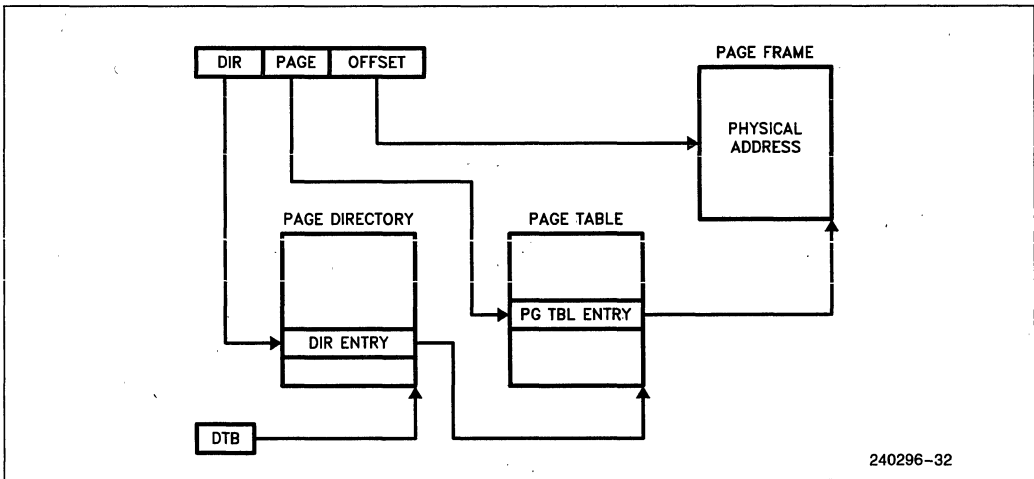


Figure 2.10. Address Translation

Two levels of tables are used to address a page of memory. At the higher level is a page directory. The page directory addresses up to 1K page tables of the second level. A page table of the second level addresses up to 1K pages. All the tables addressed by one page directory, therefore, can address 1M pages (2^{20}). Because each page contains 4 Kbytes (2^{12} bytes), the tables of one page directory can span the entire physical address space of the i860 microprocessor ($2^{20} \times 2^{12} = 2^{32}$).

The physical address of the current page directory is stored in DTB field of the **dirbase** register. Memory management software has the option of using one page directory for all processes, one page directory for each process, or some combination of the two.

2.4.4 PAGE-TABLE ENTRIES

Page-table entries (PTEs) in either level of page tables have the same format. Figure 2.11 illustrates this format.

2.4.4.1 Page Frame Address

The page frame address specifies the physical starting address of a page. Because pages are located on 4K boundaries, the low-order 12 bits are always zero. In a page directory, the page frame address is the address of a page table. In a second-level page table, the page frame address is the address of the page frame that contains the desired memory operand.

2.4.4.2 Present Bit

The P (present) bit indicates whether a page table entry can be used in address translation. P = 1 indi-

cates that the entry can be used. When P = 0 in either level of page tables, the entry is not valid for address translation, and the rest of the entry is available for software use; none of the other bits in the entry is tested by the hardware. If P = 0 in either level of page tables when an attempt is made to use a page-table entry for address translation, the processor signals either a data-access fault or an instruction-access fault. In software systems that support paged virtual memory, the trap handler can bring the required page into physical memory.

Note that there is no P bit for the page directory itself. The page directory may be not-present while the associated process is suspended, but the operating system must ensure that the page directory indicated by the **dirbase** image associated with the process is present in physical memory before the process is dispatched.

2.4.4.3 Writable and User Bits

The W (writable) and U (user) bits are used for page-level protection, which the i860 microprocessor performs at the same time as address translation. The concept of privilege for pages is implemented by assigning each page to one of two levels:

1. Supervisor level (U = 0)—for the operating system and other systems software and related data.
2. User level (U = 1)—for applications procedures and data.

The U bit of the **psr** indicates whether the i860 microprocessor is executing at user or supervisor level. The i860 microprocessor maintains the U bit of **psr** as follows:

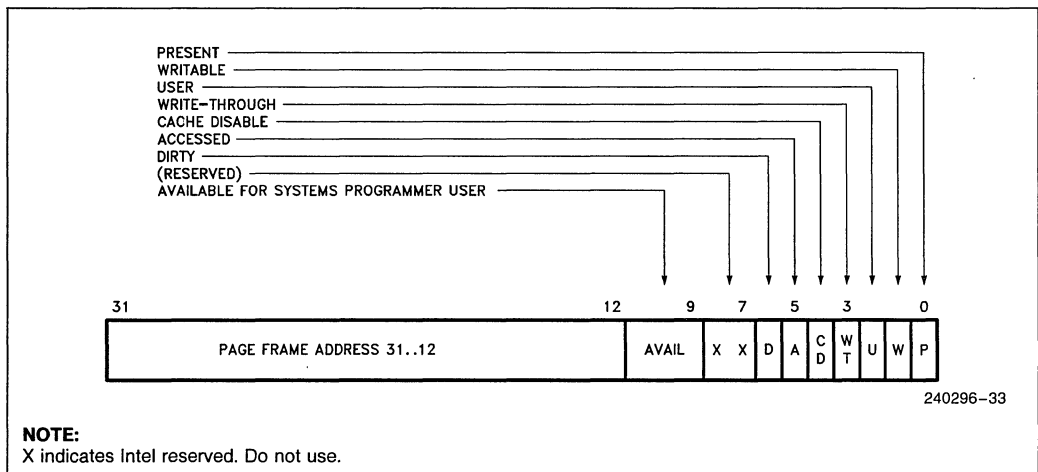


Figure 2.11. Format of a Page Table Entry

- The i860 microprocessor clears the **psr** U bit to indicate supervisor level when a trap occurs (including when the **trap** instruction causes the trap). The prior value of U is copied into PU.
- The i860 microprocessor copies the **psr** PU bit into the U bit when an indirect branch is executed and one of the trap bits is set. If PU was one, the i860 microprocessor enters user level.

With the U bit of **psr** and the W and U bits of the page table entries, the i860 microprocessor implements the following protection rules:

- When at user level, a read or write of a supervisor-level page causes a trap.
- When at user level, a write to a page whose W bit is clear causes a trap.
- When at user level, **st.c** to certain control registers is ignored.

When the i860 microprocessor is executing at supervisor level, all pages are addressable, but, when it is executing at user level, only pages that belong to the user-level are addressable.

When the i860 microprocessor is executing at supervisor level, all pages are readable. Whether a page is writable depends upon the write-protection mode controlled by WP of **epsr**:

WP = 0	All pages are writable.
WP = 1	A write to a page whose W bit is clear causes a trap.

When the i860 microprocessor is executing at user level, only pages that belong to user level and are marked writable are actually writable; pages that belong to supervisor level are neither readable nor writable from user level.

2.4.4.4 Write-Through Bit

The i860 microprocessor does not implement a write-through caching policy for the on-chip data cache; however, the WT (write-through) bit in the second-level page-table entry does determine internal caching policy. If WT is set in a PTE, on-chip caching of data from the corresponding page is inhibited. The i860 CPU may place pages having WT = 1 into the instruction cache. Future implementations of the i860 architecture may adhere to a write-through data caching policy. Therefore, they may cache pages having the WT bit of the PTE set. If WT is clear, the normal write-back policy is applied to data from the page in the on-chip caches. The WT bit of page directory entries is not referenced by the processor, but is **reserved**.

The WT bit is independent of the CD bit; therefore, data may be placed in a second-level coherent cache, but kept out of the on-chip caches.

2.4.4.5 Cache Disable Bit

If the CD (cache disable) bit in the second-level page-table entry is set, data from the associated page is not placed in instruction or data caches. Clearing CD permits the cache hardware to place data from the associated page into caches. The CD bit of page directory entries is not referenced by the processor, but is **reserved**.

To control external caches, the i860 microprocessor outputs on its PTB pin either the CD or WT bit. The PBM bit of **epsr** determines which bit is output.

2.4.4.6 Accessed and Dirty Bits

The A (accessed) and D (dirty) bits provide data about page usage in both levels of the page tables.

The i860 microprocessor sets the corresponding accessed bits in both levels of page tables before a read or write operation to a page. The processor tests the dirty bit in the second-level page table before a write to an address covered by that page table entry, and, under certain conditions, causes traps. The trap handler then has the opportunity to maintain appropriate values in the dirty bits. The dirty bit in directory entries is not tested by the i860 microprocessor. The precise algorithm for using these bits is specified in Section 2.4.5.

An operating system that supports paged virtual memory can use these bits to determine what pages to eliminate from physical memory when the demand for memory exceeds the physical memory available. The D and A bits in the PTE (page-table entry) are normally initialized to zero by the operating system. The processor sets the A bit when a page is accessed either by a read or write operation. When a data- or instruction-access fault occurs, the trap handler sets the D bit if an allowable write is being performed, then re-executes the instruction.

The operating system is responsible for coordinating its updates to the accessed and dirty bits with updates by the CPU and by other processors that may share the page tables. The i860 microprocessor automatically asserts the LOCK# signal while setting the A bit. If an A-bit of a PTE is found not set during a locked sequence (created by the **lock** instruction), a trap will occur and the processor will not update the A-bit.

2.4.4.7 Combining Protection of Both Levels of Page Tables

For any one page, the protection attributes of its page directory entry may differ from those of its page table entry. The i860 microprocessor computes the effective protection attributes for a page

by examining the protection attributes in both the directory and the page table. Table 2.6 shows the effective protection provided by the possible combinations of protection attributes.

2.4.5 ADDRESS TRANSLATION ALGORITHM

The algorithm below defines the translation of each virtual address to a physical address. Let DIR, PAGE, and OFFSET be the fields of the virtual address; let PFA1 and PFA2 be the page frame address fields of the first and second level page tables respectively; DTB is the page directory table base address stored in the **dirbase** register.

1. Read the PTE (page table entry) at the physical address formed by DTB:DIR:00.
2. If P in the PTE is zero, generate a data- or instruction-access fault.
3. If W in the PTE is zero, the operation is a write, and either the U-bit of the PSR is set or WP = 1, generate a data or instruction access fault.
4. If the U-bit in the PTE is zero and the U-bit in the **psr** is set, generate a data or instruction access fault.
5. If A in the PTE is zero, and if the TLB miss occurred while the bus was locked (due to a **lock**

instruction, or **st.c dirbase** with BL = 1), generate a data or instruction access fault. (The trap allows software to set A to one and restart the sequence. This avoids ambiguity in determining what address corresponds to a locked semaphore for external bus hardware use.)

6. If A in the PTE is zero, and if the TLB miss occurred while the bus was not locked, assert LOCK#. Re-fetch and check the PTE, set A, and store the PTE. Deassert LOCK# during the store.
7. Locate the PTE at the physical address formed by PFA1:PAGE:00.
8. Perform the P, W, U, and A checks as in steps 2 through 5 with the second-level PTE.
9. If D in the PTE is clear and the operation is a write, generate a data or instruction access fault.
10. Form the physical address as PFA2:OFFSET.

The i860 microprocessor looks only in external memory for Page Directories and Page Tables, in the translation process. The data cache is not searched. Therefore, any code which modifies Page Directories or Page Tables must keep them out of the cache. The tables should be kept in non-cacheable memory, or flushed from the cache.

Table 2.6. Combining Directory and Page Protections

Page Directory Entry		Page Table Entry		Combined Protection		
				User Access	Supervisor Access	
U-bit	W-bit	U-bit	W-bit	WP = X	WP = 0	WP = 1
0	0	0	0	N	R/W	R
0	0	0	1	N	R/W	R
0	0	1	0	N	R/W	R
0	0	1	1	N	R/W	R
0	1	0	0	N	R/W	R
0	1	0	1	N	R/W	R/W
0	1	1	0	N	R/W	R
0	1	1	1	N	R/W	R/W
1	0	0	0	N	R/W	R
1	0	0	1	N	R/W	R
1	0	1	0	R	R/W	R
1	0	1	1	R	R/W	R
1	1	0	0	N	R/W	R
1	1	0	1	N	R/W	R/W
1	1	1	0	R	R/W	R
1	1	1	1	R/W	R/W	R/W

NOTES:

N = No access allowed R/W = Both reads and writes allowed
 R = Read access only X = Don't care

The i860 microprocessor expects Page Directories and Page Tables to be in little endian format. The operating system must maintain these tables in little endian format by either setting $BE = 0$ when manipulating the tables or by complementing bit 2 of the address when loading or storing entries.

2.4.6 ADDRESS TRANSLATION FAULTS

The address translation fault is one instance of the data-access fault. The instruction causing the fault can be re-executed upon returning from the trap handler.

2.4.7 PAGE TRANSLATION CACHE

For greatest efficiency in address translation, the i860 microprocessor stores the most recently used page-table data in an on-chip cache called the TLB (translation lookaside buffer). Only if the necessary paging information is not in the cache must both levels of page tables be referenced.

2.5 Caching and Cache Flushing

The i860 microprocessor has the ability to cache instruction, data, and address-translation information in on-chip caches. Caching uses virtual-address tags. The effects of mapping two different virtual addresses in the same address space to the same physical address are undefined.

Instruction, data, and address-translation caching on the i860 microprocessor are not transparent. Writes do not immediately update memory, the TLB, nor the instruction cache. Writes to memory by other bus devices do not update the caches. Under certain circumstances, such as I/O references, self-modifying code, page-table updates, or shared data in a multi-processing system, it is necessary to bypass or to flush the caches. The i860 microprocessor provides the following methods for doing this:

- **Bypassing Instruction and Data Caches.** If deasserted during cache-miss processing, the $KEN\#$ pin disables instruction and data caching of the referenced data. If the CD bit of the associated second-level PTE is set, caching of data and instructions is disabled. The i860 CPU may place pages having $WT = 1$ into the instruction cache. Future implementations of the i860 architecture may adhere to a write-through data cache policy. Thus, they may cache pages having the WT bit of the PTE set. The value of the CD bit or the WT bit is output on the PTB pin for use by external caches.
- **Flushing Instruction and Address-Translation Caches.** Storing to the **dirbase** register with the ITI bit set invalidates the contents of the instruction and address-translation caches. This bit should be set when a page table or a page containing code is modified or when changing the DTB field of **dirbase**. Note that in order to make the instruction or address-translation caches consistent with the data cache, the data cache must be flushed *before* invalidating the other caches.

NOTE:

The mapping of the page containing the currently executing instruction and the next six instructions should not be different in the new page tables when **st.c dirbase** changes DTB or activates ITI. The six instructions following the **st.c** should be **nops** and should lie in the same page as the **st.c**.

- **Flushing the Data Cache.** The data cache is flushed by a software routine using the **flush** instruction. The data cache must be flushed prior to flushing the instruction or address-translation caches (as controlled by the ITI bit of **dirbase**) or enabling or disabling address translation (via the ATE bit). The data cache does not need flushing if the program is modifying only the P, U, W, A, or D bits of a PTE (as long as the Page Frame Address is not changed and the PTE itself was not in the data cache.) The i860 CPU does not check these protection bits on cache line writeback. Thus, a trap handler can service a DAT for D-bit-zero by setting $D = 1$ and then $ITI = 1$. In the case of setting the P or A bits active, there is no need to invalidate or flush any caches because the processor does not load entries into the TLB that have $P = 0$ or $A = 0$. The i860 microprocessor searches only external memory for Page Directories and Page Tables in the translation process. The data cache is not searched. Therefore, Page Tables and Directories should be kept in non-cacheable memory, or flushed from the cache by any code which accesses them.

2.6 Instruction Set

Table 2.7 shows the complete set of instructions grouped by function within processing unit. Refer to Section 8 for an algorithmic definition of each instruction.

The architecture of the i860 microprocessor uses parallelism to increase the rate at which operations may be introduced into the unit. Parallelism in the i860 microprocessor is **not** transparent; rather, programmers have complete control over parallelism and therefore can achieve maximum performance for a variety of computational problems.

2.6.1 PIPELINED AND SCALAR OPERATIONS

One type of parallelism used within the floating-point unit is "pipelining". The pipelined architecture treats each operation as a series of more primitive operations (called "stages") that can be executed in parallel. Consider just the floating-point adder unit as an example. Let **A** represent the operation of the adder. Let the stages be represented by **A₁**, **A₂**, and **A₃**. The stages are designed such that **A_{i+1}** for one adder instruction can execute in parallel with **A_i** for the next adder instruction. Furthermore, each **A_i** can be executed in just one clock. The pipelining within the multiplier and graphics units can be described similarly, except that the number of stages may be different.

Figure 2.12 illustrates three-stage pipelining as found in the floating-point adder (also in the floating-point multiplier when single-precision input operands are employed). The columns of the figure represent the three stages of the pipeline. Each stage holds intermediate results and also (when introduced into first stage by software) holds status information pertaining to those results. The figure assumes that the instruction stream consists of a series of consecutive floating-point instructions, all of one type (i.e. all adder instructions or all single-precision multiplier instructions). The instructions are represented as **i**, **i + 1**, etc. The rows of the figure represent the states of the unit at successive clock cycles. Each time a pipelined operation is performed, the result of the last stage of the pipeline is stored in the destination register *rdest*, the pipeline is advanced one stage, and the input operands *src1* and *src2* are transferred to the first stage of the pipeline.

In the i860 microprocessor, the number of pipeline stages ranges from one to three. A pipelined operation with a three-stage pipeline stores the result of the third prior operation. A pipelined operation with a two-stage pipeline stores the result of the second prior operation. A pipelined operation with a one-stage pipeline stores the result of the prior operation.

There are four floating-point pipelines: one for the multiplier, one for the adder, one for the graphics unit, and one for floating-point loads. The adder pipeline has three stages. The number of stages in the multiplier pipeline depends on the precision of the source operands in the pipeline. Single precision has three stages and double precision has two stages. The graphics unit has one stage for all precisions. The load pipeline has three stages for all precisions.

Changing the FZ (flush zero), RM (rounding mode), or RR (result register) bits of **fsr** while there are results in either the multiplier or adder pipeline produces effects that are not defined.

2.6.1.1 Scalar Mode

In addition to the pipelined execution mode, the i860 microprocessor also can execute floating-point instructions in "scalar" mode. Most floating-point instructions have both pipelined and scalar variants, distinguished by a bit in the instruction encoding. In scalar mode, the floating-point unit does not start a new operation until the previous floating-point operation is completed. The scalar operation passes through all stages of its pipeline before a new operation is introduced, and the result is stored automatically. Scalar mode is used when the next operation depends on results from the previous few floating-point operations (or when the compiler or programmer does not want to deal with pipelining).

2.6.1.2 Pipelining Status Information

Result status information in the **fsr** consists of the AA, AI, AO, AU, and AE bits, in the case of the adder, and the MA, MI, MO, and MU bits, in the case of the multiplier. This information arrives at the **fsr** via the pipeline in one of two ways:

Table 2.7. Instruction Set

Core Unit		Floating-Point Unit	
Mnemonic	Description	Mnemonic	Description
Load and Store Instructions		F-P Multiplier Instruction	
ld.x	Load integer	fmul.p	F-P multiply
st.x	Store integer	pfmul.p	Pipelined F-P multiply
fld.y	F-P load	pfmul3.dd	3-Stage pipelined F-P multiply
pfld.z	Pipelined F-P load	fmlow.p	F-P multiply low
fst.y	F-P store	frcp.p	F-P reciprocal
pst.d	Pixel store	frsqr.p	F-P reciprocal square root
Register to Register Moves		F-P Adder Instructions	
ixfr	Transfer integer to F-P register	fadd.p	F-P add
fxfr	Transfer F-P to integer register	pfadd.p	Pipelined F-P add
Integer Arithmetic Instructions		famov.r	F-P adder move
addu	Add unsigned	pfamov.r	Pipelined F-P adder move
adds	Add signed	fsub.p	F-P subtract
subu	Subtract unsigned	pfsub.p	Pipelined F-P subtract
subs	Subtract signed	pfgt.p	Pipelined F-P greater-than compare
Shift Instructions		pfreq.p	Pipelined F-P equal compare
shl	Shift left	fix.p	F-P to integer conversion
shr	Shift right	pfix.p	Pipelined F-P to integer conversion
shra	Shift right arithmetic	fttrunc.p	F-P to integer truncation
shrd	Shift right double	pftrunc.p	Pipelined F-P to integer truncation
Logical Instructions		Dual-Operation Instructions	
and	Logical AND	pfam.p	Pipelined F-P add and multiply
andh	Logical AND high	pfsm.p	Pipelined F-P subtract and multiply
andnot	Logical AND NOT	pfmam.p	Pipelined F-P multiply with add
andnoth	Logical AND NOT high	pfmsm.p	Pipelined F-P multiply with subtract
or	Logical OR	Long Integer Instructions	
orh	Logical OR high	fisub.z	Long-integer subtract
xor	Logical exclusive OR	pfisub.z	Pipelined long-integer subtract
xorh	Logical exclusive OR high	fiadd.z	Long-integer add
Control-Transfer Instructions		pfisub.z	Pipelined long-integer add
trap	Software trap	Graphics Instructions	
intovr	Software trap on integer overflow	fzchks	16-bit Z-buffer check
br	Branch direct	pfzchks	Pipelined 16-bit Z-buffer check
bri	Branch indirect	fzchkl	32-bit Z-buffer check
bc	Branch on CC	pfzchkl	Pipelined 32-bit Z-buffer check
bc.t	Branch on CC taken	faddp	Add with pixel merge
bnc	Branch on not CC	pfaddp	Pipelined add with pixel merge
bnc.t	Branch on not CC taken	faddz	Add with Z merge
bte	Branch if equal	pfaddz	Pipelined add with Z merge
btrne	Branch if not equal	form	OR with MERGE register
bla	Branch on LCC and add	pforn	Pipelined OR with MERGE register
call	Subroutine call	Assembler Pseudo-Operations	
calli	Indirect subroutine call	Mnemonic	
System Control Instructions		Description	
flush	Cache flush	mov	Integer register-register move
ld.c	Load from control register	fmov.r	F-P reg-reg move
st.c	Store to control register	pfmov.r	Pipelined F-P reg-reg move
lock	Begin interlocked sequence	nop	Core no-operation
unlock	End interlocked sequence	fnop	F-P no-operation
		pfle.p	Pipelined F-P less-than or equal

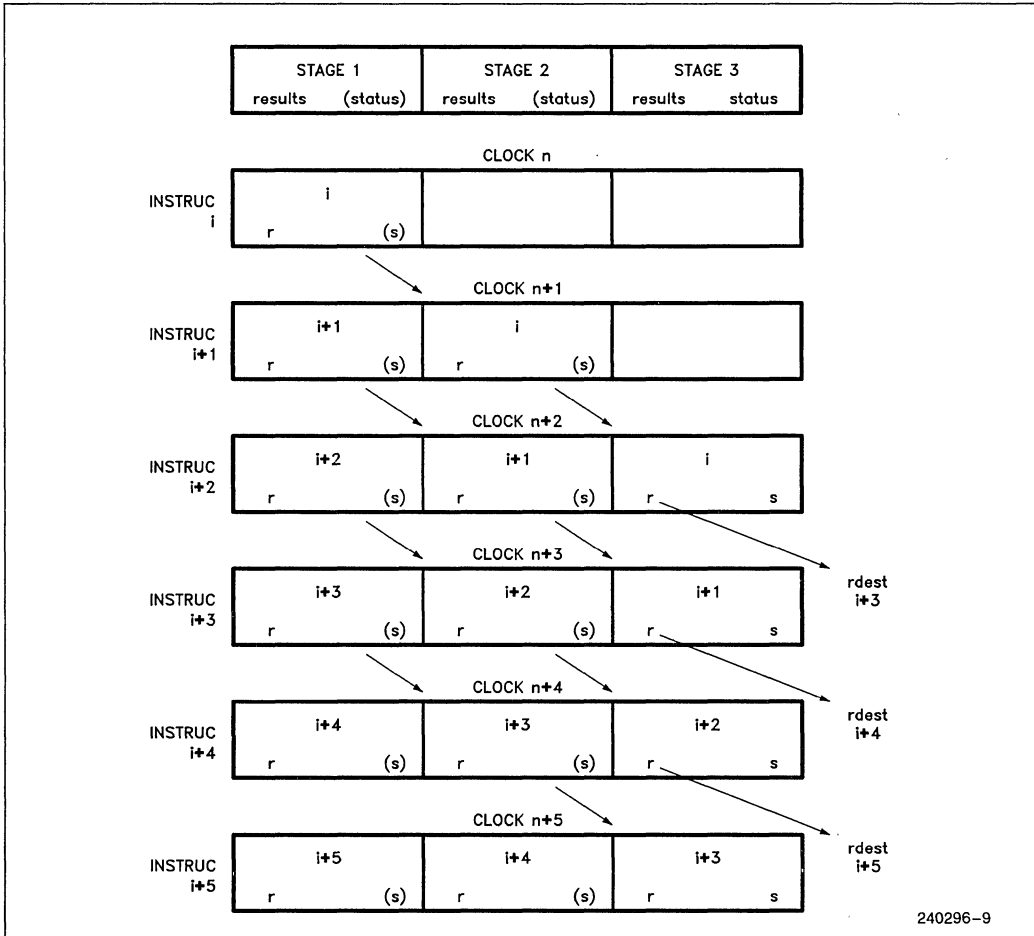


Figure 2.12. Pipelined Instruction Execution

1. It is calculated by the last stage of the pipeline. This is the normal case.
2. It is propagated from the first stage of the pipeline. This method is used when restoring the state of the pipeline after a preemption. When a store instruction updates the **fsr** and the value of the U bit in the word being written into the **fsr** is set, the store updates the result status bits in the first stage of both the adder and multiplier pipelines. When software changes the result-status bits of the first stage of a particular unit (multiplier or adder), the updated result-status bits are propagated one stage for each pipelined floating-point operation for that unit. In this case, each stage of the adder and multiplier pipelines holds its own copy of the relevant bits of the **fsr**. When they reach the last stage, they override the normal result-status bits computed from the last stage result.

At the next floating-point instruction (or at certain core instructions), after the result reaches the last stage, the i860 microprocessor traps if any of the status bits of the **fsr** indicate exceptions. Note that the instruction that creates the exceptional condition is not the instruction at which the trap occurs.

2.6.1.3 Precision in the Pipelines

In pipelined mode, when a floating-point operation is initiated, the result of an earlier pipelined floating-point operation is returned. The result precision of the current instruction applies to the operation being initiated. The precision of the value stored in **rdest** is that which was specified by the instruction that initiated that operation.

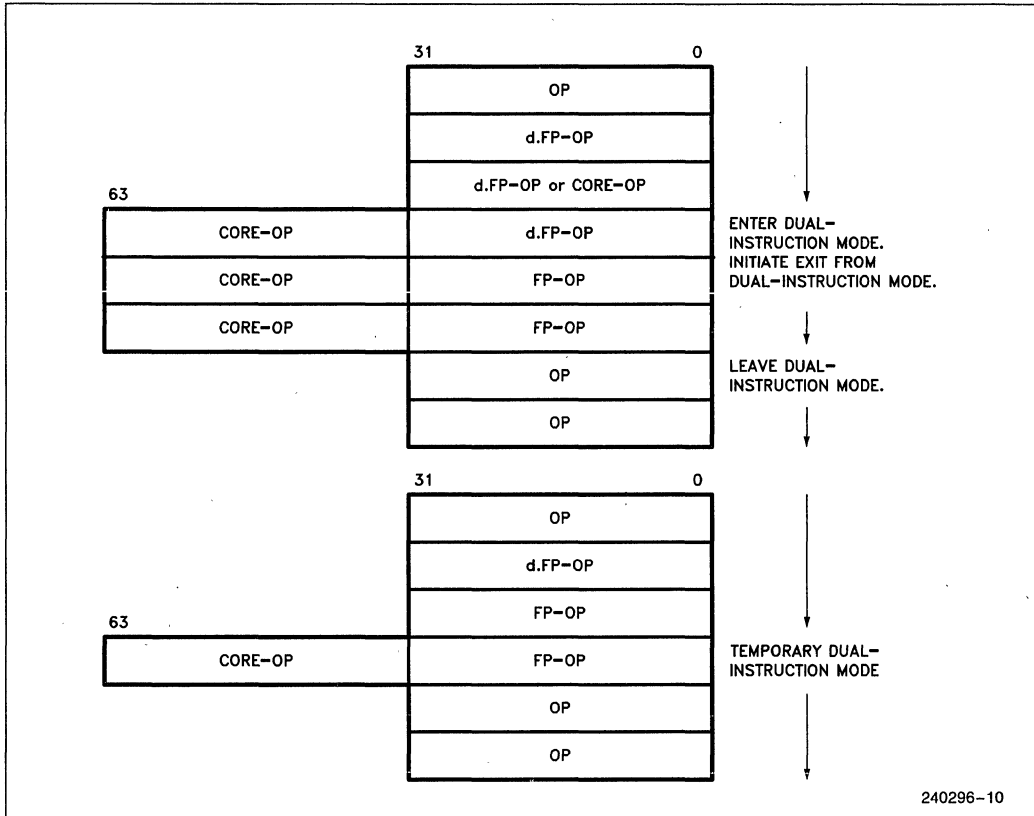


Figure 2.13. Dual-Instruction Mode Transitions

If *rdest* is the same as *src1* or *src2*, the value being stored in *rdest* is used as the input operand. In this case, the precision of *rdest* must be the same as the source precision.

The multiplier pipeline has two stages when the source operand is double-precision and three stages when the precision of the source operand is single. This means that a pipelined multiplier operation stores the result of the second previous multiplier operation for double-precision inputs and third previous for single-precision inputs (except when changing precisions).

2.6.1.4 Transition between Scalar and Pipelined Operations

When a scalar operation is executed, it passes through all stages of the pipeline; therefore, any unstored results in the affected pipeline are lost. To avoid losing information, the last pipelined operations before a scalar operation should be dummy pipelined operations that unload unstored results from the affected pipeline.

After a scalar operation, the values of all pipeline stages of the affected unit (except the last) are undefined. No spurious result-exception traps result when the undefined values are subsequently stored by pipelined operations; however, the values should not be referenced as source operands.

For best performance a scalar operation should not immediately precede a pipelined operation whose *rdest* is nonzero.

2.6.2 DUAL-INSTRUCTION MODE

Another form of parallelism results from the fact that the i860 microprocessor can execute both a floating-point and a core instruction simultaneously. Such parallel execution is called dual-instruction mode. When executing in dual-instruction mode, the instruction sequence consists of 64-bit aligned instructions with a floating-point instruction in the lower 32 bits and a core instruction in the upper 32 bits. Table 2.7 identifies which instructions are executed by the core unit and which by the floating-point unit.

Programmers specify dual-instruction mode either by including in the mnemonic of a floating-point instruction a **d**. prefix or by using the Assembler directives **.dualenddual**. Both of the specifications cause the D-bit of floating-point instructions to be set. If the i860 microprocessor is executing in single-instruction mode and encounters a floating-point instruction with the D-bit set, one more 32-bit instruction is executed before dual-mode execution begins. If the i860 microprocessor is executing in dual-instruction mode and a floating-point instruction is encountered with a clear D-bit, then one more pair of instructions is executed before resuming single-instruction mode. Figure 2.13 illustrates two variations of this sequence of events: one for extended sequences of dual-instructions and one for a single instruction pair.

When a 64-bit dual-instruction pair sequentially follows a delayed branch instruction in dual-instruction mode, both 32-bit instructions are executed.

2.6.3 DUAL-OPERATION INSTRUCTIONS

Special dual-operation floating-point instructions (add-and-multiply, subtract-and-multiply) use both the multiplier and adder units within the floating-point unit in parallel to efficiently execute such common tasks as evaluating systems of linear equations, performing the Fast Fourier Transform (FFT), and performing graphics transformations.

The instructions **pfam src1, src2, rdest** (add and multiply), **pfsm src1, src2, rdest** (subtract and multiply), **pfmam src1, src2, rdest** (multiply and add), and **pfmsm src1, src2, rdest** (multiply and subtract) initiate both an adder operation and a multiplier operation. Six operands are required, but the instruction format specifies only three operands; therefore, there are special provisions for specifying the operands. These special provisions consist of:

- Three special registers (KR, KI, and T), that can store values from one dual-operation instruction and supply them as inputs to subsequent dual-operation instructions.
 1. The constant registers KR and KI can store the value of *src1* and subsequently supply that value to the multiplier pipeline in place of *src1*.
 2. The transfer register T can store the last stage result of the multiplier pipeline and subsequently supply that value to the adder pipeline in place of *src1*.
- A four-bit data-path control field in the opcode (DPC) that specifies the operands and loading of the special registers.
 1. Operand-1 of the multiplier can be KR, KI, or *src1*.
 2. Operand-2 of the multiplier can be *src2* or the last stage result of the adder pipeline.

3. Operand-1 of the adder can be *src1*, the T-register, or the last stage result of the adder pipeline.
4. Operand-2 of the adder can be *src2*, the last stage result of the multiplier pipeline, or the last stage result of the adder pipeline.

Figure 2.14 shows all the possible data paths surrounding the adder and multiplier. A DPC field in these instructions select different data paths. Section 8 shows the various encodings of the DPC field.

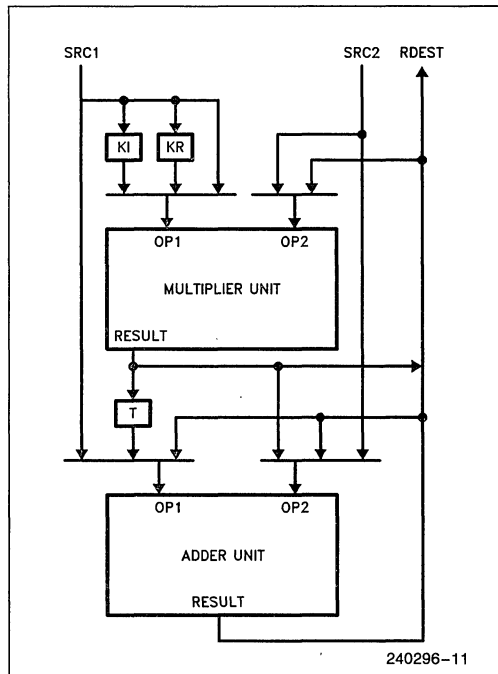


Figure 2.14. Dual-Operation Data Paths

Note that the mnemonics **pfam.p**, **pfsm.p**, **pfmam.p**, and **pfmsm.p** are never used as such in the assembly language; these mnemonics are used here to designate classes of related instructions. Each value of DPC has a unique mnemonic associated with it.

2.7 Addressing Modes

Data access is limited to load and store instructions. Memory addresses are computed from two fields of load and store instructions: *src1* and *src2*.

1. *src1* either contains the identifier of a 32-bit integer register or contains an immediate 16-bit address offset.
2. *src2* always specifies a register.

Table 2.8. Types of Traps

Type	Indication		Caused by	
	PSR,ESPR	FSR	Condition	Instruction
Instruction Fault	IT OF IL		Software traps Missing unlock	trap , intovr Any
Floating Point Fault	FT	SE AO, MO AU, MU Ai, Mi	Floating-point source exception Floating-point result exception overflow underflow inexact result	Any M- or A-unit except fmflow Any M- or A-unit except fmflow , ptgt , and pfeq . Reported on any F-P instruction plus pst , fst , and sometimes fid , pfid , ixfr
Instruction Access Fault	IAT		Address translation exception during instruction fetch	Any
Data Access Fault	DAT*		Load/store address translation exception Misaligned operand address Operand address matches db register	Any load/store Any load/store Any load/store
Interrupt	IN		External interrupt	
Reset	No trap bits set		Hardware RESET signal	

NOTES:

*These cases can be distinguished by examining the operand addresses.

The IL bit of the **psr** must be checked by the trap handler to tell if the bus is currently in a locked sequence.

Because either *src1* or *src2* may be null (zero), a variety of useful addressing modes result:

offset + register Useful for accessing fields within a record, where *register* points to the beginning of the record. Useful for accessing items in a stack frame, where *register* is *r3*, the register used for pointing to the beginning of the stack frame.

register + register Useful for two-dimensional arrays or for array access within the stack frame.

register Useful as the end result of any arbitrary address calculation.

offset Absolute address into the first or last 32K of the logical address space.

In addition, the floating-point load and store instructions may select autoincrement addressing. In this mode *src2* is replaced by the sum of *src1* and *src2* after performing the load or store. This mode makes stepping through arrays more efficient, because it eliminates one address-calculation instruction.

2.8 Traps and Interrupts

Traps are caused by exceptional conditions detected in programs or by external interrupts. Traps

cause interruption of normal program flow to execute a special program known as a trap handler. Traps are divided into the types shown in Table 2.8. Interrupts and traps start execution in single instruction mode at virtual address 0xFFFFF00 in supervisor level (U = 0).

2.8.1 TRAP HANDLER INVOCATION

This section applies to traps other than reset. When a trap occurs, execution of the current instruction is aborted. The instruction is restartable. The processor takes the following steps while transferring control to the trap handler:

1. Copies U (user mode) of the **psr** into PU (previous U).
2. Copies IM (interrupt mode) into PIM (previous IM).
3. Sets U to zero (supervisor mode).
4. Sets IM to zero (interrupts disabled).
5. If the processor is in dual instruction mode, it sets DIM; otherwise it clears DIM.
6. If the processor is in single-instruction mode and the next instruction will be executed in dual-instruction mode or if the processor is in dual-instruction mode and the next instruction will be executed in single-instruction mode, DS is set; otherwise, it is cleared.

7. The appropriate trap type bits in **psr** are set (IT, IN, IAT, DAT, FT). Several bits may be set if the corresponding trap conditions occur simultaneously.
8. An address is placed in the fault instruction register (**fir**) to help locate the trapped instruction. In single-instruction mode, the address in **fir** is the address of the trapped instruction itself. In dual-instruction mode, the address in **fir** is that of the floating-point half of the dual instruction. If an instruction or data access fault occurred, the associated core instruction is the high-order half of the dual instruction (**fir** + 4). In dual-instruction mode, when a data access fault occurs in the absence of other trap conditions, the floating-point half of the dual instruction will already have been executed.

The processor begins executing the trap handler by transferring execution to virtual address 0xFFFFF00. The trap handler begins execution in single-instruction mode. The trap handler must examine the trap-type bits in **psr** (IT, IN, IAT, DAT, FT) to determine the cause or causes of the trap.

2.8.2 INSTRUCTION FAULT

This fault is caused by any of the following conditions. In all cases the processor sets the IT bit before entering the trap handler.

- By the **trap** instruction.
- By the **intovr** instruction. The trap occurs only if OF in **epsr** is set when **intovr** is executed. The trap handler should clear OF before returning.
- By the lack of an **unlock** instruction within 32 instructions of a **lock**. In this case IL is also set. When the trap handler finds IL set, it should scan backwards for the **lock** instruction and restart at that point. The absence of a **lock** instruction within 32 instructions of the trap indicates a programming error.

2.8.3 FLOATING-POINT FAULT

The floating-point fault occurs on floating-point instructions, **pst**, **fst**, and sometimes **fld**, **pfld**, **ixfr**. The floating-point faults of the i860 microprocessor support the floating-point exceptions defined by the IEEE standard as well as some other useful classes of exceptions. The i860 microprocessor divides these into two classes: source exceptions and result exceptions. The numerics library supplied by Intel provides the IEEE standard default handling for all these exceptions.

2.8.3.1 Source Exception Faults

When used as inputs to the multiplier or adder, all exceptional operands, including infinities, denormal-

ized numbers and NaNs, cause a floating-point fault and set SE in the **fsr**. Source exceptions are reported on the instruction that initiates the operation. For pipelined operations, the pipeline is not advanced.

The SE value is undefined for faults on **fld**, **pfld**, **fst**, **pst**, and **ixfr** instructions when in single-instruction mode or when in dual-instruction mode and the companion instruction is not a multiplier or adder operation.

2.8.3.2 Result Exception Faults

The class of result exceptions includes any of the following conditions:

- **Overflow**. The absolute value of the rounded true result would exceed the largest positive finite number in the destination format.
- **Underflow** (when FZ is clear). The absolute value of the rounded true result would be smaller than the smallest positive finite number in the destination format.
- **Inexact result** (when TI is set). The result is not exactly representable in the destination format. For example, the fraction $\frac{1}{3}$ cannot be precisely represented in binary form. This exception occurs frequently and indicates that some (generally acceptable) accuracy has been lost.

The point at which a result exception is reported depends upon whether pipelined operations are being used:

- **Scalar (nonpipelined) operations**. Result exceptions are reported on the next floating-point, **fst.x**, or **pst.x** (and sometimes **fld**, **pfld**, **ixfr**) instruction after the scalar operation. When a trap occurs, the last stage of the affected unit contains the result of the scalar operation.
- **Pipelined operations**. Result exceptions are reported when the result is in the last stage and the next floating-point, **fst.x** or **pst.x** (and sometimes **fld**, **pfld**, **ixfr**) instruction is executed. When a trap occurs, the pipeline is not advanced, and the last stage results (that caused the trap) remain unchanged.

When no trap occurs (either because FTE is clear or because no exception occurred), the pipeline is advanced normally by the new floating-point operation. The result-status bits of the affected unit are undefined until the point that result exceptions are reported. At this point, the last stage result-status bits (bits 29..22 and 16..9 of the **fsr**) reflect the values in the last stages of both the adder and multiplier. For example, if the last stage result in the multiplier has overflowed and a pipelined floating-point **pfadd** is started, a trap occurs and MO is set.

For scalar operations, the RR bits of **fsr** specify the register in which the result was stored. RR is updated when the scalar instruction is initiated. The trap, however, occurs on a subsequent instruction. Programmers must prevent intervening stores to **fsr** from modifying the RR bits. Prevention may take one of the following forms:

- Before any store to **fsr** when a result exception may be pending, execute a dummy floating-point operation to trigger the result-exception trap.
- Always read from **fsr** before storing to it, and mask updates so that the RR bits are not changed.

For pipelined operations, RR is cleared and the result is in the last stage of the pipeline of the appropriate unit. The trap handler must flush the pipeline, saving the results and the status bits.

In either pipelined or scalar mode, the trap handler must then compute the trapping result. In either case, the result has the same fraction as the true result and has an exponent which is the low-order bits of the true result. The trap handler can inspect the result, compute the result appropriate for that instruction (a NaN or an infinity, for example), and store the correct result. The result is either stored in the register specified by RR (if nonzero) or (if RR = 0) the trap handler must reload the pipeline with the saved results and status bits. The trap handler must clear the result status for the last stage and then reexecute the trapping instruction.

Result exceptions may be reported for both the adder and multiplier units at the same time. In this case, the trap handler should fix up the last stage of both pipelines.

2.8.4 INSTRUCTION ACCESS FAULT

This trap results from a page-not-present exception during instruction fetch. If a supervisor-level page is fetched in user mode, an exception may or may not occur.

2.8.5 DATA ACCESS FAULT

This trap results from an abnormal condition detected during data operand fetch or store. Such an exception can be due only to one of the following causes:

- An attempt is being made to write to a page whose D (Dirty) bit is clear.
- A memory operand is misaligned (is not located at an address that is a multiple of the length of the data).
- The address stored in the **db** register is equal to one of the addresses spanned by the operand.
- The operand is in a not-present page.

- An attempt is being made from user level to write to a read-only page or to access a supervisor-level page.
- The operand was in a page whose PTE had A = 0, and the access occurred during a locked sequence. (i.e., between **lock** and **unlock**.)
- Write protection (determined by **epsr** bit WP = 1) is violated in supervisor mode.

2.8.6 INTERRUPT TRAP

An interrupt is an event that is signaled from an external source. If the processor is executing with interrupts enabled (IM set in the **psr**), the processor sets the interrupt bit IN in the **psr**, and generates an interrupt trap. Vectored interrupts are implemented by interrupt controllers and software.

2.8.7 RESET TRAP

When the i860 microprocessor is reset, execution begins in single-instruction mode at physical address 0xFFFFF00. This is the same address as for other traps. The reset trap can be distinguished from other traps by the fact that no trap bits are set. The instruction cache is flushed. The bits DPS, BL, and ATE in **dirbase** are cleared. CS8 is initialized by the value at the INT pin at the end of reset. The read-only fields of the **espr** are set to identify the processor, while the IL, WP, and PBM bits are cleared. The bits U, IM, BR, and BW in **psr** are cleared. All other bits of **psr** and all other register contents are **undefined**.

The software must ensure that the data cache is flushed and control registers are properly initialized before performing operations that depend on the values of the cache or registers. The data cache has no "validity" bits, so memory accesses before the flush may result in false data cache hits.

Reset code must initialize the floating-point pipeline state to zero with floating-point traps disabled to ensure that no spurious floating-point traps are generated.

After a RESET the i860 microprocessor starts execution at supervisor level (U=0). Before branching to the first user-level instruction, the RESET trap handler or subsequent initialization code has to set PU and a trap bit so that an indirect branch instruction will copy PU to U, thereby changing to user level.

2.9 Debugging

The i860 microprocessor supports debugging with both data and instruction breakpoints. The features

of the i860 architecture that support debugging include:

- **db** (data breakpoint register) which permits specification of a data addresses that the i860 microprocessor will monitor.
- **BR** (break read) and **BW** (break write) bits of the **psr**, which enable trapping of either reads or writes (respectively) to the address in **db**.
- **DAT** (data access trap) bit of the **psr**, which allows the trap handler to determine when a data breakpoint was the cause of the trap.
- **trap** instruction that can be used to set breakpoints in code. Any number of code breakpoints can be set. The values of the *src1* and *src2* fields help identify which breakpoint has occurred.
- **IT** (instruction trap) bit of the **psr**, which allows the trap handler to determine when a **trap** instruction was the cause of the trap.

3.0 HARDWARE INTERFACE

In the following description of hardware interface, the # symbol at the end of a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no # is present after the signal name, the signal is asserted when at the high voltage level.

3.1 Signal Description

Table 3.1 identifies functional groupings of the pins, lists every pin by its identifier, gives a brief description of its function, and lists some of its characteristics. All output pins are tristate, except HLDA and BREQ. All inputs are synchronous, except HOLD and INT.

3.1.1 CLOCK (CLK)

The CLK input determines execution rate and timing of the i860 microprocessor. Timing of other signals is specified relative to the rising edge of this signal. The i860 microprocessor can utilize a clock rate of 33.3 MHz or 40 MHz. The internal operating frequency is the same as the external clock. This signal is TTL compatible.

3.1.2 SYSTEM RESET (RESET)

Asserting RESET for at least 16 CLK periods causes initialization of the i860 microprocessor. Refer to section 3.2 "Initialization" for more details related to RESET.

3.1.3 BUS HOLD (HOLD) AND BUS HOLD ACKNOWLEDGE (HLDA)

These pins are used for i860 microprocessor bus arbitration. At some clock after the HOLD signal is

asserted, the i860 microprocessor releases control of the local bus and puts all bus interface outputs (except BREQ and HLDA) into a floating state, then asserts HLDA—all during the same clock period. It maintains this state until HOLD is deasserted. Instruction execution stops only if required instructions or data cannot be read from the on-chip instruction and data caches.

The time required to acknowledge a hold request is one clock plus the number of clocks needed to finish any outstanding bus cycles. HOLD is recognized even while RESET or LOCK# are asserted.

When leaving a bus hold, the i860 microprocessor deactivates HLDA and, in the same clock period, initiates a pending bus cycle, if any.

Hold is an asynchronous input.

3.1.4 BUS REQUEST (BREQ)

This signal is asserted when the i860 microprocessor has a pending memory request, even when HLDA is asserted. This allows an external bus arbiter to implement an "on demand only" policy for granting the bus to the i860 microprocessor. BREQ is asserted the clock after the i860 microprocessor realizes an internal request for the bus. BREQ is deasserted with the last ADS# for which there is a pending request.

3.1.5 INTERRUPT/CODE-SIZE (INT/CS8)

This input allows interruption of the current instruction stream. If interrupts are enabled (IM set in **psr**) when INT is asserted, the i860 microprocessor fetches the next instruction from address 0xFFFFF00. To assure that an interrupt is recognized, INT should remain asserted until the software acknowledges the interrupt (by writing, for example, to a memory-mapped port of an interrupt controller). The maximum time between the assertion of INT and execution of the first instruction of the trap handler is 10 clocks, plus the time for twenty nonpipelined read cycles (six TLB misses, with four refetches when the A bit is zero) plus the time for eight nonpipelined writes (updates to the A bit), plus the time for four sets of four pipelined read cycles and two sets of four pipelined writes (instruction and data cache misses and write-back cycles to update memory) for non-locked sequences.

If the bus is locked from a **lock** instruction, the INT pin is ignored and the INT bit of **epsr** is always zero. The **lock** instruction can only assert LOCK# for 30–33 clock cycles before trapping. If the bus is locked from a **st.c dirbase** with BL = 1 then interrupts are fully functional and the timing of service is the same as for non-locked sequences.

Table 3.1. Pin Summary

Pin Name	Function	Active State	Input/Output
Execution Control Pins			
CLK	CLock		I
RESET	System reset	High	I
HOLD	Bus hold	High	I
HLDA	Bus hold acknowledge	High	O
BREQ	Bus request	High	O
INT/CS8	Interrupt, code-size	High	I
Bus Interface Pins			
A31–A3	Address bus	High	O
BE7#–BE0#	Byte Enables	Low	O
D63–D0	Data bus	High	I/O
LOCK#	Bus lock	Low	O
W/R#	Write/Read bus cycle	Hi/Low	O
NENE#	NEAr	Low	O
NA#	Next Address request	Low	I
READY#	Transfer Acknowledge	Low	I
ADS#	ADdress Status	Low	O
Cache Interface Pins			
KEN#	Cache ENable	Low	I
PTB	Page Table Bit	High	O
Testability Pins			
SHI	Boundary Scan Shift Input	High	I
BSCN	Boundary Scan Enable	High	I
SCAN	Shift Scan Path	High	I
Intel-Reserved Configuration Pins			
CC1–CC0	Configuration	High	I
Power and Ground Pins			
V _{CC}	System power		
V _{SS}	System ground		

A # after a pin name indicates that the signal is active when at the low voltage level.

If INT is asserted during the clock before the falling edge of RESET, the eight-bit code-size mode is selected. For more about this mode, refer to section 3.2 “Initialization”.

INT is an asynchronous input.

3.1.6 ADDRESS PINS (A31–A3) AND BYTE ENABLES (BE7#–BE0#)

The 29-bit address bus (A31–A3) identifies addresses to a 64-bit location. Separate byte-enable signals (BE7#–BE0#) identify which bytes should be accessed within the 64-bit location. Cache reads return 64 bits without regard for the byte-enable signals.

Instruction fetches (W/R# is low) are distinguished from data accesses by the unique combinations of

BE7#–BE0# defined in Table 3.2. For an eight-bit code fetch in eight-bit code-size (CS8) mode, BE2#–BE0# are redefined to be A2–A0 of the address. In this case BE7#–BE3# form the code shown in Table 3.2 that identifies an instruction fetch. The A2 in the table does not represent a physical pin, just a conceptual internal address line value. The “x” under A2 for CS8 mode means “not applicable”, or “don’t care”. All other combinations of byte enables indicate data accesses.

3.1.7 DATA PINS (D63–D0)

The bus interface has 64 bidirectional data pins (D63–D0) to transfer data in eight- to 64-bit quantities. Pins D7–D0 transfer the least significant byte; pins D63–D56 transfer the most significant byte.

In write bus cycles, the point at which data is driven onto the bus depends on the type of the preceding cycle. If there was no preceding cycle (i.e. the bus was idle), data is driven with the address. If the preceding cycle was a write, data is driven as soon as **READY#** is returned from the previous cycle. If the preceding cycle was a read, data is driven one clock after **READY#** is returned from the previous cycle, thereby allowing time for the bus to be turned around.

3.1.8 BUS LOCK (LOCK#)

This signal is used to provide atomic (indivisible) read-modify-write sequences in multiprocessor systems. A multiprocessor bus arbiter must permit only one processor a locked access to the address which is on the bus when **LOCK#** first activates. The system must maintain the lock of that location until **LOCK#** deactivates.

The i860 microprocessor coordinates the external **LOCK#** signal with the software-controlled **BL** bit of the **dirbase** register. Programmers do not have to be concerned about the fact that bus activity is not always synchronous with instruction execution. **LOCK#** is asserted with **ADS#** for the first bus cycle that results from an instruction executed after the **BL** bit is set. Pending bus cycles are locked according to the **BL** bit when the instruction was executed. Even if the **BL** bit is changed between the time that an instruction generates an internal bus request and the time that the cycle appears on the bus, the i860 microprocessor still asserts **LOCK#** for that bus cycle.

If **ADS#** is active when **LOCK#** deactivates, then that request should complete before the hardware relinquishes the lock. If **ADS#** is not active, the locking of the location can immediately end when **LOCK#** deactivates. Of course the simplest arbitration hardware can just lock the entire bus against all other accesses during **LOCK#** assertion.

The deassertion of **LOCK#** depends on how the **BL** bit was deasserted. If the **BL** bit is deasserted with the **unlock** instruction, **LOCK#** is deasserted with the next **ld** or **st** but after any pending bus cycles. Between locked sequences, at least one cycle of

no **LOCK#** is guaranteed by the behavior of the **unlock** instruction. **LOCK#** deassertion may occur independently of **ADS#** for the case of a trap or a cache hit after **unlock**.

If **BL** is deasserted with a **st.c dirbase** instruction, **LOCK#** is only deasserted for a following **ld** or **st** instruction that causes a cache miss. **LOCK#** stays asserted for pending bus cycles and intervening cache hits.

The i860 microprocessor also asserts **LOCK#** during TLB miss processing for updates of the accessed bit in page-table entries. The maximum time that **LOCK#** can be asserted in this case is five clocks plus the time required by software to perform a read-modify-write sequence. Instruction fetches do not alter the **LOCK#** pin.

Between **lock** and **unlock** instructions, the **INT** pin is ignored and the **INT** bit of **epsr** is always zero when read by **ld.c epsr**.

3.1.9 WRITE/READ BUS CYCLE (W/R#)

This pin specifies whether a bus cycle is a read (**LOW**) or write (**HIGH**) cycle.

3.1.10 NEXT NEAR (NENE#)

This signal allows higher-speed reads and writes in the case of consecutive reads and writes that access static column or page-mode DRAMs. The i860 microprocessor asserts **NENE#** when the current address is in the same DRAM page as the previous bus cycle. The i860 microprocessor determines the DRAM page size by inspecting the **DPS** field in the **dirbase** register. The page size can range from 2⁹ to 2¹⁶ 64-bit words, supporting DRAM sizes from 256K × 1, 256K × 4, and up. **NENE#** is never asserted on the next bus cycle after **HLDA** is deasserted.

3.1.11 NEXT ADDRESS REQUEST (NA#)

NA# makes address pipelining possible. The system asserts **NA#** to indicate that it is ready to ac-

Table 3.2. Identifying Instruction Fetches

Code Fetch	A2	BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#
Normal (Non-CS8)	0	1	1	1	1	1	0	1	0
Normal (Non-CS8)	1	1	0	1	0	1	1	1	1
CS8 Mode	x	1	0	1	0	1	Low-order address bits		

cept the next address from the i860 microprocessor. NA# may be asserted before the current cycle ends. (If the system does not implement pipelining, NA# does not have to be activated.) The i860 microprocessor samples NA# every clock, starting one clock after the prior activation of ADS#. When NA# is active, the i860 microprocessor is free to drive address and bus-cycle definition for the next pending bus cycle. The i860 microprocessor remembers that NA# was asserted when no internal request is pending; therefore, NA# can be deactivated after the next rising edge of the CLK signal. Up to three bus cycles can be outstanding simultaneously.

3.1.12 TRANSFER ACKNOWLEDGE (READY#)

The system asserts the READY# signal during read cycles when valid data is on the data pins and during a write cycles when the system has accepted data from the data pins. READY# is sampled one clock after prior ADS# or prior READY# in case of pipelining.

3.1.13 ADDRESS STATUS (ADS#)

The i860 microprocessor asserts ADS# during the first clock of each bus cycle to identify the clock period during which it begins to assert outputs on the address bus. This signal is not held active during a pipelined bus cycle. This allows two-level pipelining, for a maximum of three outstanding cycles.

3.1.14 CACHE ENABLE (KEN#)

The i860 microprocessor samples KEN# to determine whether the data being read for the current cache-miss cycle is to be cached. This pin is internally NORed with the CD and WT bits to control cacheability on a page by page basis (refer to Table 3.3).

If the address is one that is permitted to be in the cache, KEN# must be continuously asserted during the sampling period starting from the clock after ADS# is asserted, through the clock NA# or READY# is asserted. The entire 64 bits of the data bus will be used for the read, regardless of the state of the byte-enable pins. Three additional 64-bit bus cycles will be generated to fill the rest of the 32-byte cache block.

If KEN# is found deasserted at any clock from the clock after ADS# through the clock of the first NA# or READY#, the data being read will not be cached and two scenarios can occur: 1) if the cycle is due to data-cache miss, no subsequent cache-fill cycles will be generated; 2) if the cycle is due to an instruction-cache miss, additional cycle(s) will be generated until the address reaches a 32-byte boundary. To avoid caching a line, external hardware must de-

assert KEN# during or before the first NA# or READY#.

3.1.15 PAGE TABLE BIT (PTB)

Depending on the setting of the PBM (page-table bit mode) bit of the **epsr**, the PTB reflects the value of either the CD (cache disable) bit or the WT (write through) bit of the page-table entry used for the current cycle. When paging is disabled, PTB remains inactive.

Table 3.3. Cacheability based on KEN# and CD OR'ed WT

CD OR'ed WT	KEN#	Meaning
0	0	Cacheable access
0	1	Noncacheable access
1	0	Noncacheable page
1	1	Noncacheable page

3.1.16 BOUNDARY SCAN SHIFT INPUT (SHI)

This pin is used with the testability features. Refer to section 3.3.

3.1.17 BOUNDARY SCAN ENABLE (BSCN)

This pin is used with the testability features. Refer to section 3.3.

3.1.18 SHIFT SCAN PATH (SCAN)

This pin is used with the testability features. Refer to section 3.3.

3.1.19 CONFIGURATION (CC1–CC0)

These two pins are reserved by Intel. Strap both pins LOW.

3.1.20 SYSTEM POWER (V_{CC}) AND GROUND (V_{SS})

The i860 microprocessor has 48 pins for power and ground. All pins must be connected to the appropriate low-inductance power and ground signals in the system.

3.2 Initialization

Initialization of the i860 microprocessor is caused by assertion of the RESET signal for at least 16 clocks. Table 3.4 shows the status of output pins during the time that RESET is asserted. Note that HOLD requests are honored during RESET and that the status of output pins depends on whether a HOLD request is being acknowledged.

Table 3.4. Output Pin Status during Reset

Pin Name	Pin Value	
	HOLD Not Acknowledged	HOLD Acknowledged
ADS#, LOCK#	HIGH	Tri-State OFF
W/R#, PTB	LOW	Tri-State OFF
BREQ	LOW	LOW
HLDA	LOW	HIGH
D63-D0	Tri-State OFF	Tri-State OFF
A31-A3, BE7#-BE0#, NENE#	Undefined	Tri-State OFF

After a reset, the i860 microprocessor begins executing at physical address 0xFFFFF00. The program-visible state of the i860 microprocessor after reset is detailed in section 2.8.7.

Eight-bit code-size mode is selected when INT/CS8 is asserted during the clock before the falling edge of RESET. While in eight-bit code-size mode, instruction cache misses are byte reads (transferred on D7-D0 of the data bus) instead of eight-byte reads. This allows the i860 microprocessor to be bootstrapped from an eight-bit EPROM. For these code reads, byte enables BE2#-BE0# are redefined to be the low order three bits of the address, so that a complete byte address is available. These reads update the instruction cache if KEN# is asserted (refer to section 3.1.14) and are not pipelined even if NA# is asserted. While in this mode, instructions must reside in an eight-bit wide memory, while data must reside in a separate 64-bit wide memory. After the code has been loaded into 64-bit memory, initialization code can initiate 64-bit code fetches by clearing the CS8 bit of the **dirbase** register (refer to section 2). Once eight-bit code-size mode is disabled by software, it cannot be reenabled except by resetting the i860 microprocessor.

3.3 Testability

The i860 microprocessor has a *boundary scan mode* that may be used in component- or board-level testing to test the signal traces leading to and from the i860 microprocessor. Boundary scan mode provides a simple serial interface that makes it possible to test all signal traces with only a few probes. Probes need be connected only to CLK, BSCN, SCAN, SHI, BREQ, RESET, and HOLD.

The pins BSCN and SCAN control the boundary scan mode (refer to Table 3.5). When BSCN is as-

serted, the i860 microprocessor enters boundary scan mode on the next rising clock edge. Boundary scan mode can be activated even while RESET is active. When BSCN is deasserted while in boundary scan mode, the i860 microprocessor leaves boundary scan mode on the next rising clock edge. After leaving boundary scan mode, the internal state is undefined; therefore, RESET should be asserted.

Table 3.5. Test Mode Selection

BSCN	SCAN	Testability Mode
LO	LO	No testability mode selected (Reserved for Intel)
LO	HI	Boundary scan mode, normal
HI	LO	Boundary scan mode, shift
HI	HI	SHI as input; BREQ as output

For testing purposes, each signal pin has associated with it an internal latch. Table 3.6 identifies these latches by name and classifies them as input, output, or control. The input and output latches carry the name of the corresponding pins.

Table 3.6. Test Mode Latches

Input Latch	Output Latch	Associated Control Latch
SHI		
BSCN		
SCAN		
RESET		
D0-D63	D0-D63	DATA _t
CC1-CC0		
	A31-A3	ADDR _t
	NENE#	NENEt
	PTB#	PTB _t
	W/R#	W/R _t
	ADS#	ADSt
	HLDA	
	LOCK#	LOCK _t
READY#		
KEN#		
NA#		
INT/CS8		
HOLD		
	BE7#-BE0#	BE _t
	BREQ	

Within boundary scan mode the i860 microprocessor operates in one of two submodes: normal mode or shift mode, depending on the value of the SCAN input. A typical test sequence is . . .

1. Enter shift mode to assign values to the latches that correspond with the pins.
2. Enter normal mode. In normal mode the i860 microprocessor transfers the latched values to the output pins and latches the values that are being driven onto the input pins.
3. Reenter shift mode to read the new values of the input pins.

3.3.1 NORMAL MODE

When SCAN is deasserted, the normal mode is selected. For each input pin (RESET, HOLD, INT/CS8, NA#, READY#, KEN#, SHI, BSCN, SCAN, CC1, and CC0), the corresponding latch is loaded with the value that is being driven onto the pin.

The tristate output pins (A31–A3, BE7#–BE0#, W/R#, NENE#, ADS#, LOCK#, and PTB) are enabled by the control latches ADDRt (for A31–A3), BEt, W/Rt, NENEt, ADSt, LOCKt, and PTBt. If a control latch is set, the corresponding output latches drive their output pins; otherwise the pins are not driven.

The I/O pins (D63–D0) are enabled by the control latch DATAt, which is similar to the other control latches. In addition, when DATAt is not set, the data pins are treated as input pins and their values are latched.

3.3.2 SHIFT MODE

When SCAN is asserted, the shift mode is selected. In shift mode, the pins are organized into a *boundary scan chain*. The scan chain is configured as a shift register that is shifted on the rising edge of CLK. The SHI pin is connected to the input of one end of the boundary scan chain. The value of the most significant bit of the scan chain is output on the BREQ pin. To avoid glitches while the values are being shifted along the chain, the tester should assert both the RESET and HOLD pins. Then all tristate outputs are disabled. The order of the pins within the chain is shown in Figure 3.1.

A tester causes entry into this mode for one of two purposes:

1. To assign values to output latches to be driven onto output pins upon subsequent entry into normal mode.
2. To read the values of input pins previously latched in normal mode.

4.0 BUS OPERATION

A bus cycle begins when ADS# is activated and ends when READY# is sampled active. READY# is sampled one clock after assertion of ADS# and thereafter until it becomes active. New cycles can start as often as every other clock until three cycles are outstanding. A bus cycle is considered outstanding as long as READY# has not been asserted to terminate that cycle. After READY# becomes active, it is not sampled again for the following (outstanding) cycle until the second clock after the one during which it became active. READY# is assumed to be inactive when it is not sampled.

With regard to how a bus cycle is generated by the i860 microprocessor, there are two types of cycles: pipelined and nonpipelined. Both types of cycles can be either read or write cycles. A pipelined cycle is one that starts while one or two other bus cycles are outstanding. A nonpipelined cycle is one that starts when no other bus cycles are outstanding.

4.1 Pipelining

A *m-n* read or write cycle is a cycle with a total cycle time of *m* clocks and a cycle-to-cycle time of *n* clocks ($m \geq n$). Total cycle time extends from the clock in which ADS# is activated to the clock in which READY# becomes active, whereas, cycle-to-cycle time extends from the time that READY# is sampled active for the previous cycle to the time that it is sampled active again for the current cycle. When $m = n$, a nonpipelined cycle is implied; $m > n$ implies a pipelined cycle.

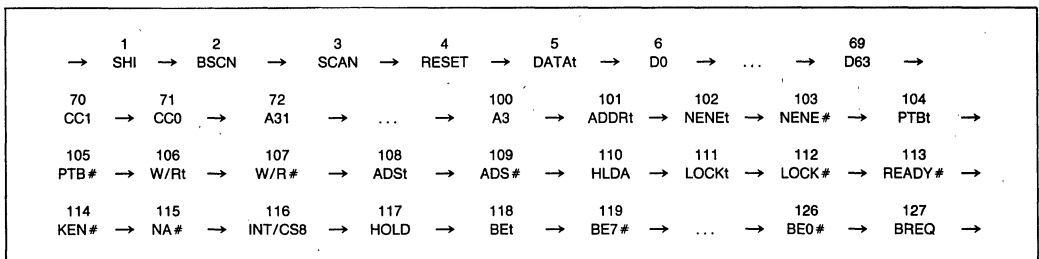


Figure 3.1. Order of Boundary Scan Chain

Pipelining may occur for the next bus cycle any time the current bus cycle requires more than two clock periods to finish ($m > 2$). The next cycle can be initiated when $NA\#$ is sampled active, even if the current cycle has not terminated. In this case, pipelining occurs. $NA\#$ is recognized only in the clock when $ADS\#$ has become inactive.

To allow high transfer rates in large memory systems, two-level pipelining is supported (i.e., there may be up to three cycles in progress at one time). Pipelining enables a new word of data to be transferred every two clocks, even though the total cycle time may be up to six clocks.

4.2 Bus State Machine

The operation of the bus is described in terms of a bus state machine using a state transition diagram. Figure 4.1 illustrates the i860 microprocessor bus state machine. A bus cycle is composed of two or more states. Each bus state lasts for one CLK period.

The i860 microprocessor supports up to two levels of address pipelining. Once it has started the first bus cycle, it can generate up to two more cycles as long as $READY\#$ remains inactive. To start a new bus cycle while other cycles are still outstanding, $NA\#$ must be active for at least one clock cycle starting with the clock after the previous $ADS\#$. $NA\#$ is latched internally.

States T_j and T_{jk} , for $j = \{1,2,3\}$ and $k = \{1,2\}$, are used to describe the state of the i860 microprocessor Bus State Machine. Index j indicates the number of outstanding bus cycles while index k distinguishes the intermediate states for the j -th outstanding cycle.

Therefore there can be up to three outstanding cycles, and there are two possible intermediate states for each level of pipelining. T_{j1} is the next state after T_j , as long as j cycles are outstanding. T_{j2} is entered when $NA\#$ is active but the i860 microprocessor is not ready to start a new cycle.

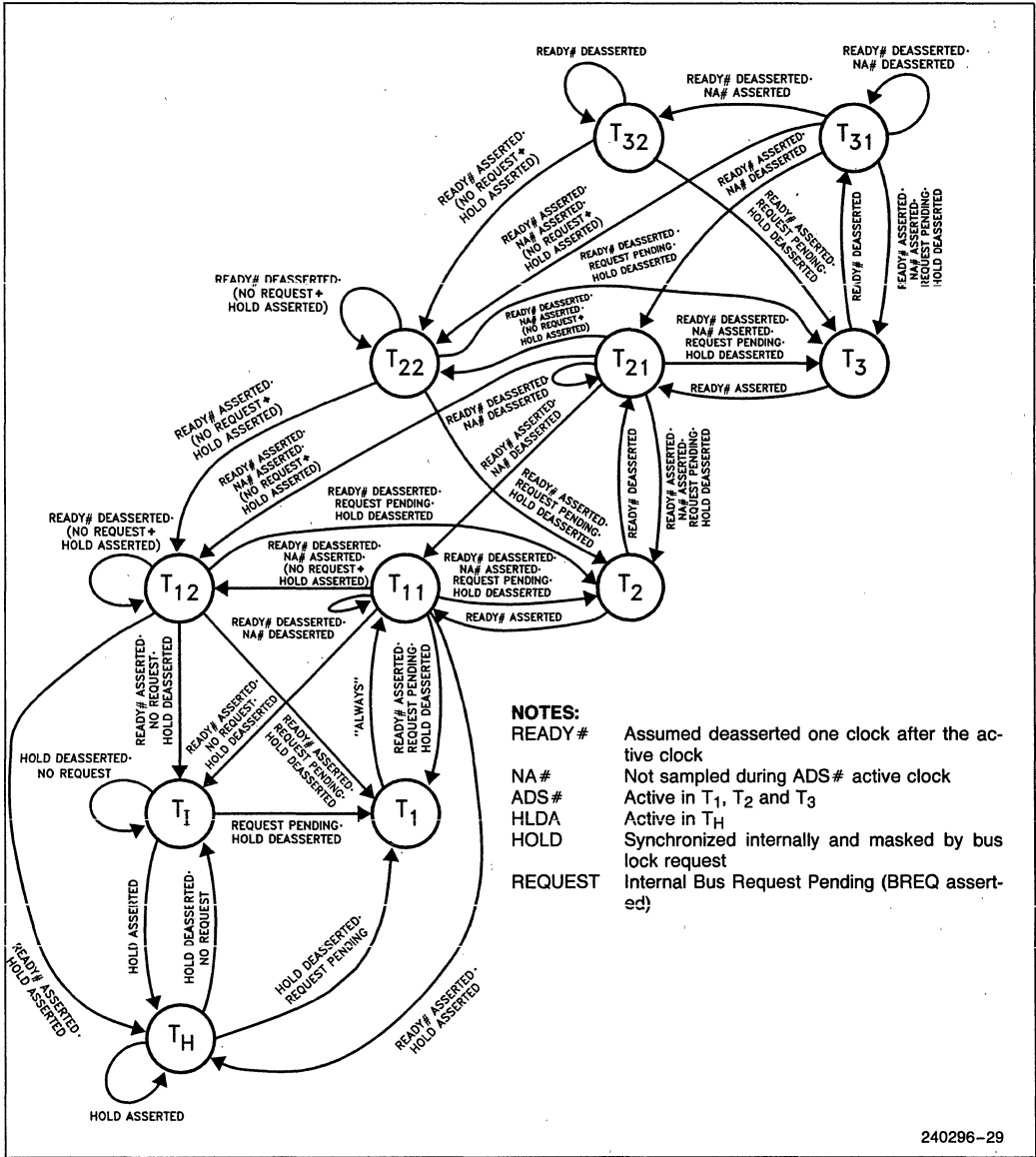
Five conditions have to be met to start a new cycle while one or more cycles are already pending:

1. $READY\#$ inactive
2. $NA\#$ having been active
3. An internal request pending ($BREQ$ active)
4. $HOLD$ not active
5. Fewer than three cycles outstanding

Note that $BREQ$ is asserted on the clock after the i860 microprocessor realizes an internal request for the bus.

Upon hardware $RESET$, the bus control logic enters the idle state T_1 and awaits an internal request for a bus cycle. If a bus cycle is requested while there is no hold request from the system, a bus cycle begins, advancing to state T_1 . On the next cycle, the state machine automatically advances to state T_{11} . If $READY\#$ is active in state T_{11} , the bus control logic returns either to T_1 , if no new cycle is started, or to T_1 , if a new cycle request is pending internally. In fact, if an internal bus request is pending each time $READY\#$ is active, the state machine continues to cycle between T_{11} and T_1 .

However, if $READY\#$ is not active but the next address request is pending (as indicated by an active $NA\#$), the state machine advances either to state T_2 (if an internal bus request is pending, signifying that two bus cycles are now outstanding), or to state T_{12} (if no bus internal request is pending, signifying $NA\#$ has been found active). Transitions from state T_{12} are similar to those from T_{11} .



240296-29

Figure 4.1. Bus State Machine

If two bus cycles are already outstanding (as indicated by T_{2k} for k = {1,2}) and NA# is latched active but READY# is not active, one more bus request causes entry into state T₃. Transitions from this state are similar to those from T₂.

machine continues to oscillate between T_{j1} and T_j, for j = {2,3}.

In general, if there is an internal bus request each time both READY# and NA# are active, the state

When NA# is sampled active while there is a pending bus request, ADS# is activated in the next clock period (provided no more than two cycles are already outstanding).

Internal pending bus requests start new bus cycles only if no HOLD request has been recognized. T_H is entered from the idle state T_1 , T_{11} , and T_{12} . HLDA is active in this state. There is a one clock delay to synchronize the HOLD input when the signal meets the respective minimum setup and hold time requirements. The state machine uses the synchronized HOLD to move from state to state.

4.3 Bus Cycles

Figures 4.2 through 4.10 illustrate combinations of bus cycles.

4.3.1 NONPIPELINED READ CYCLES

A read cycle begins with the clock in which $ADS\#$ is asserted. The i860 microprocessor begins driving the address during this clock. It samples $READY\#$ for active state every clock after the first clock. A minimum of two clocks is required per cycle. Data is latched when $READY\#$ is found active when sampled at the end of a clock period. Figure 4.2 illustrates nonpipelined read cycles with zero wait states.

Normally, all 64 bits of the data bus are latched; however, in the case of noncacheable bus cycles, the byte enables $BE7\#-BE0\#$ determine which bytes are used. In CS8 mode, only the low-order eight bits are latched.

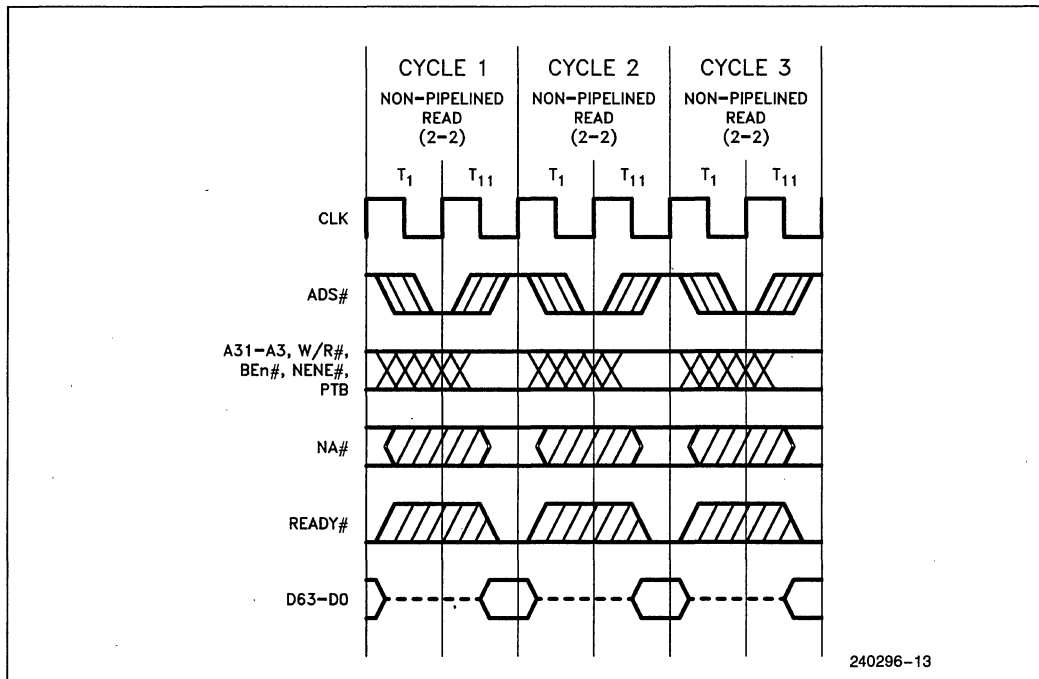


Figure 4.2. Fastest Read Cycles

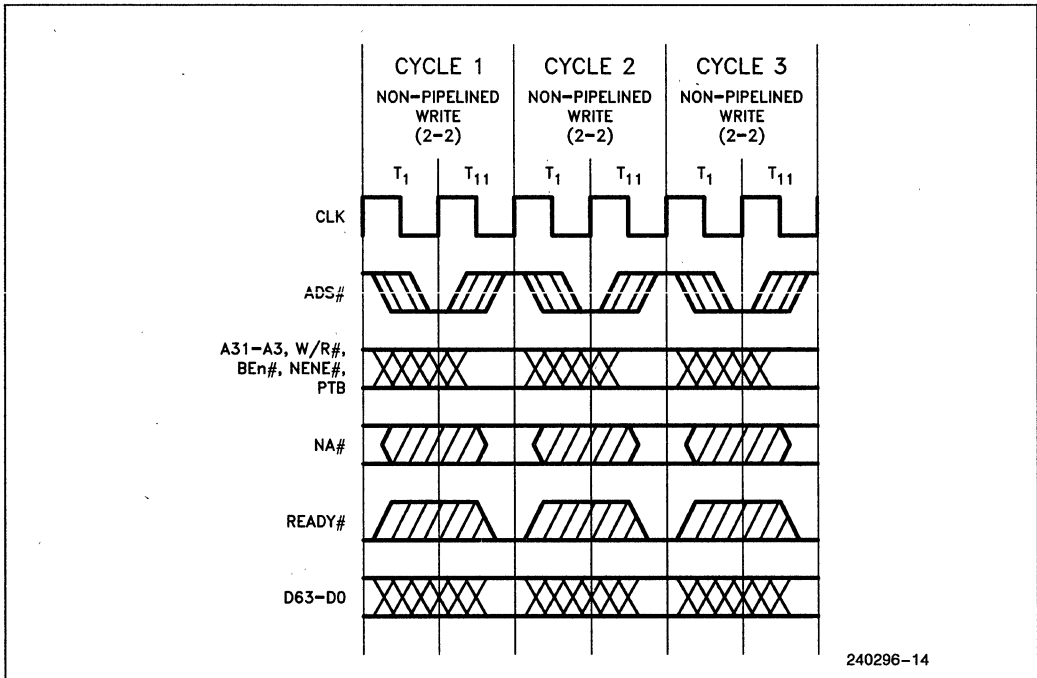


Figure 4.3. Fastest Write Cycles

4.3.2 NONPIPELINED WRITE CYCLES

The ADS# and READY# activity for write cycles follows the same logic as that for read cycles, as Figure 4.3 illustrates for back-to-back, nonpipelined write cycles with zero wait-states. The byte enables BE7#-BE0# indicate which bytes on the data bus are valid.

The fastest write cycle takes only two clocks to complete. However, when a read cycle immediately pre-

cedes a write cycle, the write cycle must contain a wait state, as illustrated in Figure 4.4. Because the device being read might still be driving the data bus during the first clock of the write cycle, there is a potential for bus contention. To help avoid such contention, the i860 microprocessor does not drive the data bus until the second clock of the write cycle. The wait state is required to provide the additional time necessary to terminate the write cycle. In other read-write combinations, the i860 microprocessor does not require a wait state.

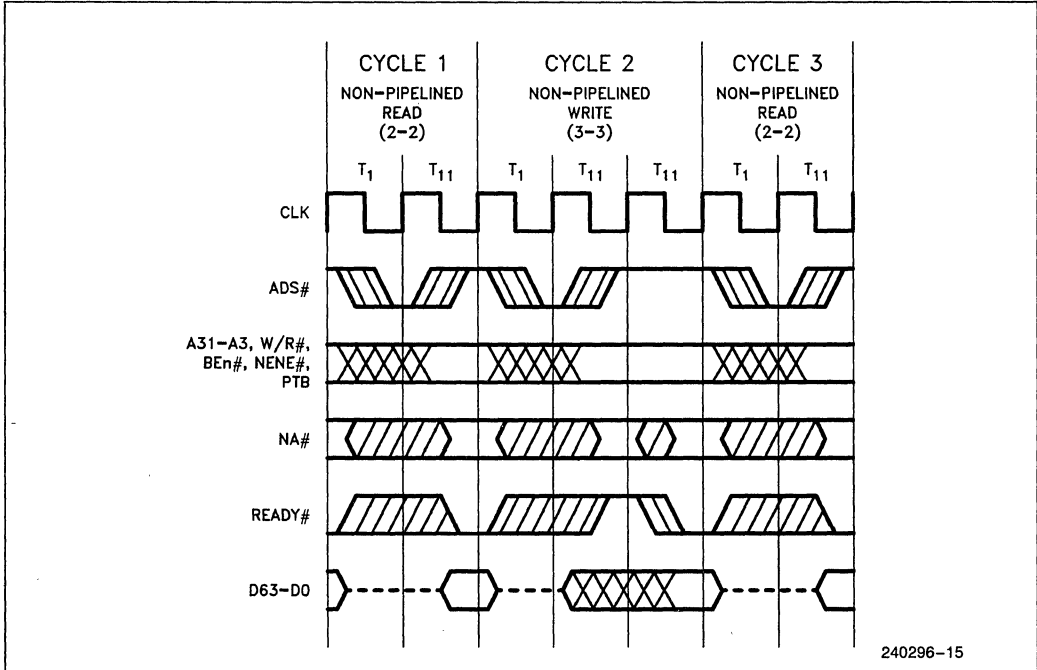


Figure 4.4. Fastest Read/Write Cycles

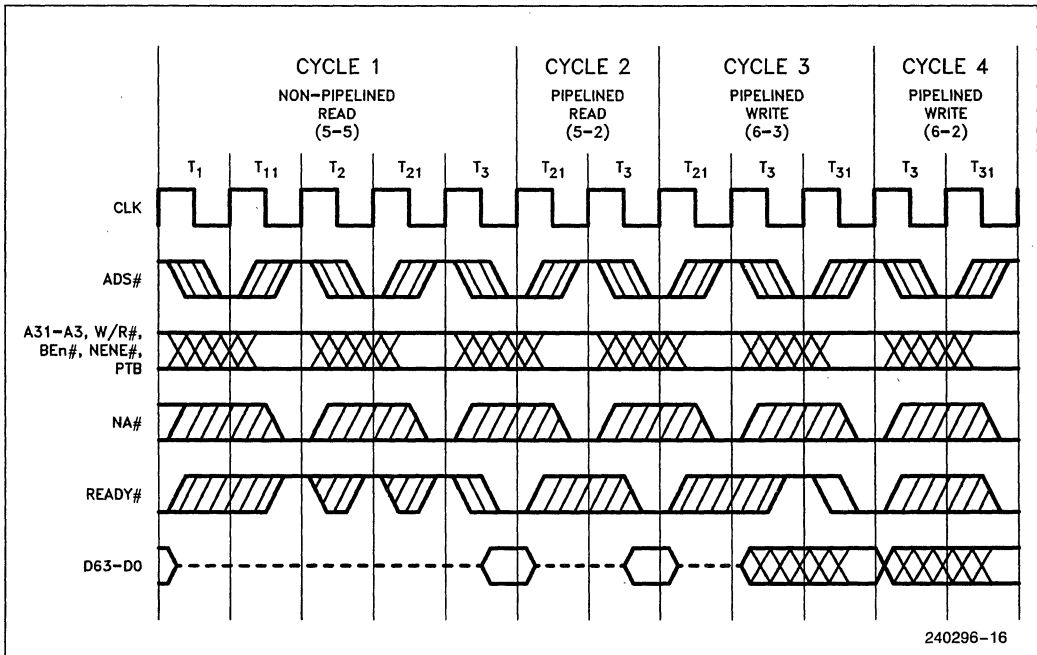


Figure 4.5. Pipelined Read Followed by Pipelined Write

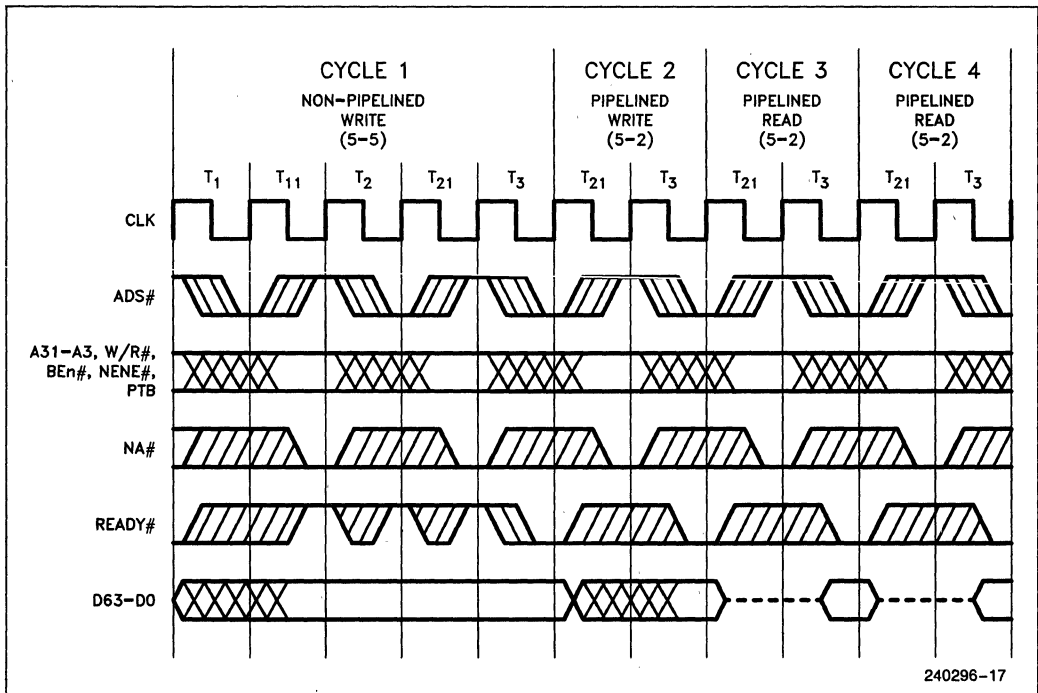


Figure 4.6. Pipelined Write Followed by Pipelined Read

4.3.3 PIPELINED READ AND WRITE CYCLES

Figures 4.5 and 4.6 illustrate combinations of non-pipelined and pipelined read and write cycles. The following description applies to both diagrams. While Cycle 1 is still in progress, two new cycles are initiated. By the time READY# first becomes active, the state machine has moved through states T₁, T₁₁, T₂, T₂₁, and T₃. Cycles 3 and 4 show how activating READY# terminates the corresponding outstanding cycle, and yet activating NA# while there is an internal request pending adds a new outstanding cycle.

In Figure 4.5, Cycle 3 is a write cycle following a read cycle; therefore, one wait state must be inserted. The i860 microprocessor does not drive the data bus until one clock after the read data is returned from the preceding read cycle. During Cycles 3 and 4, the state machine oscillates between states T₃

and T₃₁ maintaining full bus capacity (two levels of pipelining; three outstanding cycles). Cycles 2, 3, and 4 in Figure 4.6 are 5-2 cycles; i.e. each requires a total cycle time of five clocks while the throughput rate is one cycle every two clocks.

Figure 4.7 illustrates in a more general manner how the NA# signal controls pipelining. Cycle 1 is a 2-2 cycle, the fastest possible. The next cycle cannot be started any earlier; therefore, there is no need to activate NA# to start the next cycle early. Cycle 2, a 3-3 read, is different. Cycle 3 can be started during the third state (a wait state) of Cycle 2, and NA# is asserted to accomplish this.

NA# is not activated following the ADS# clock of Cycle 3, thereby allowing Cycle 3 to terminate before the start of Cycle 4. As a result, Cycle 4 is a nonpipelined cycle.

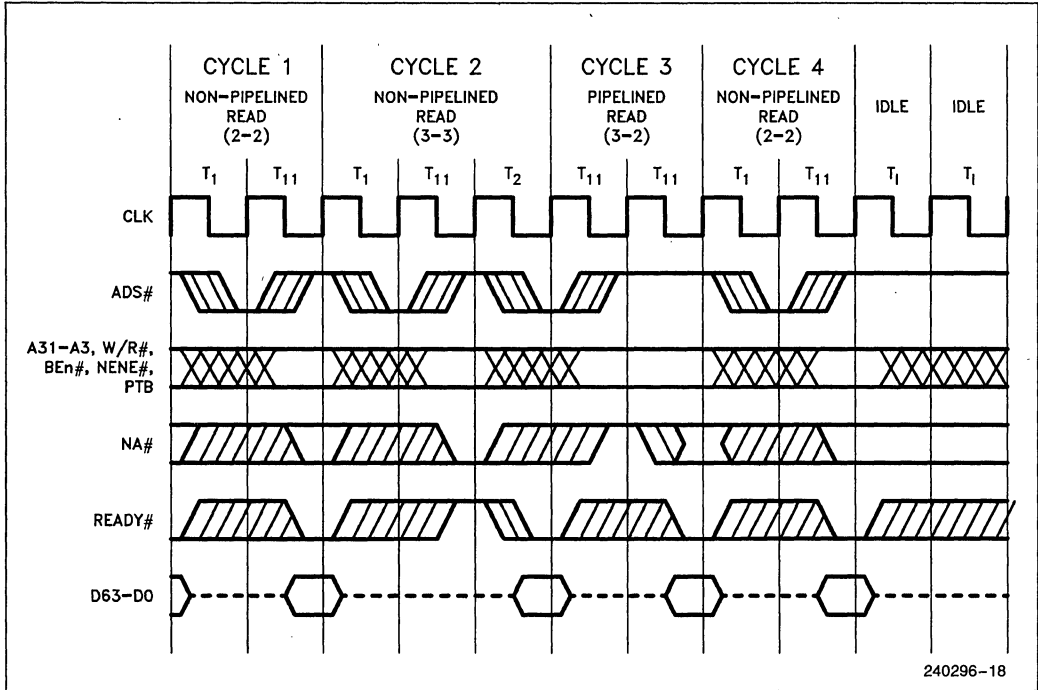


Figure 4.7. Pipelining Driven by NA #

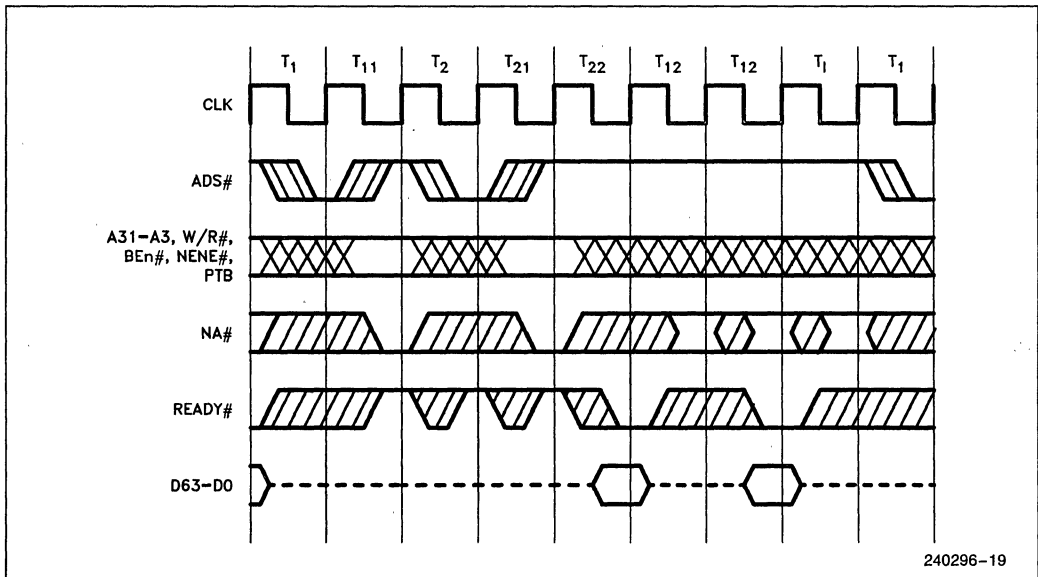


Figure 4.8. NA # Active with No Internal Bus Request

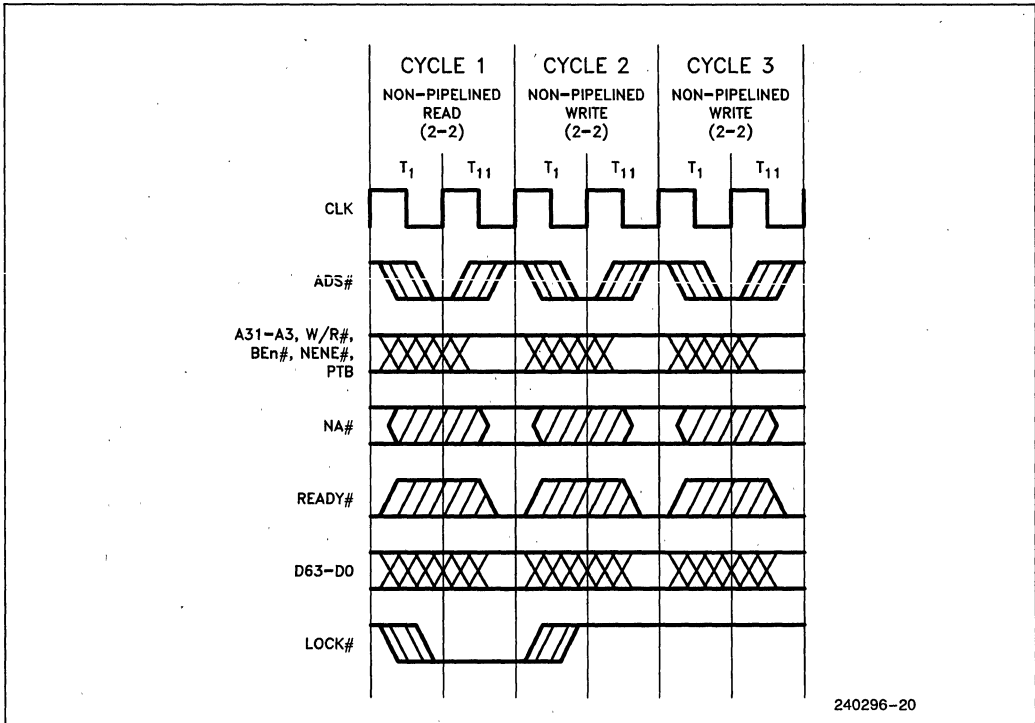


Figure 4.9. Locked Cycles

240296-20

When there is no internal bus request, activating NA# does not start a new cycle; the i860 microprocessor, however, remembers that NA# has been activated. Figure 4.8 illustrates the situation where NA# is active but no internal bus request is pending. Because there is no internal request pending until after one idle state, no new bus cycle is started during that period.

4.3.4 LOCKED CYCLES

The LOCK# signal is asserted when the current bus cycle is to be locked with the next bus cycle. Assertion of LOCK# may be initiated by a program's setting the BL bit of the **dirbase** register using the **lock** instruction or **st.c dirbase** with BL = 1 (refer to section 2) or by the i860 microprocessor itself during page table updates.

In Figure 4.9, the first read cycle is to be locked with the following write cycle. If there were idle states between the cycles, the LOCK# signal would remain asserted. This is the case for a read/modify/write operation. The second write cycle is not locked

because LOCK# is no longer asserted when the first write cycle starts.

4.3.5 HOLD AND BREQ ARBITRATION CYCLES

The HOLD, HLDA, and BREQ signals permit bus arbitration between the i860 microprocessor and another bus master.

See Figure 4.10. When HOLD is asserted, the i860 microprocessor does not relinquish control of the bus until all outstanding cycles are completed. If HOLD were asserted one clock earlier, the last i860 microprocessor bus cycle before HLDA would not be started.

The outputs (except HLDA and BREQ) float when HLDA is asserted. HOLD is sampled at the end of the clock in which it is activated. Recommended setup and hold times must be met to guarantee sampling one clock after external HOLD activation. When HOLD is sampled active, a one clock delay for internal synchronization follows. HLDA may be deasserted as early as the clock following deassertion of HOLD.

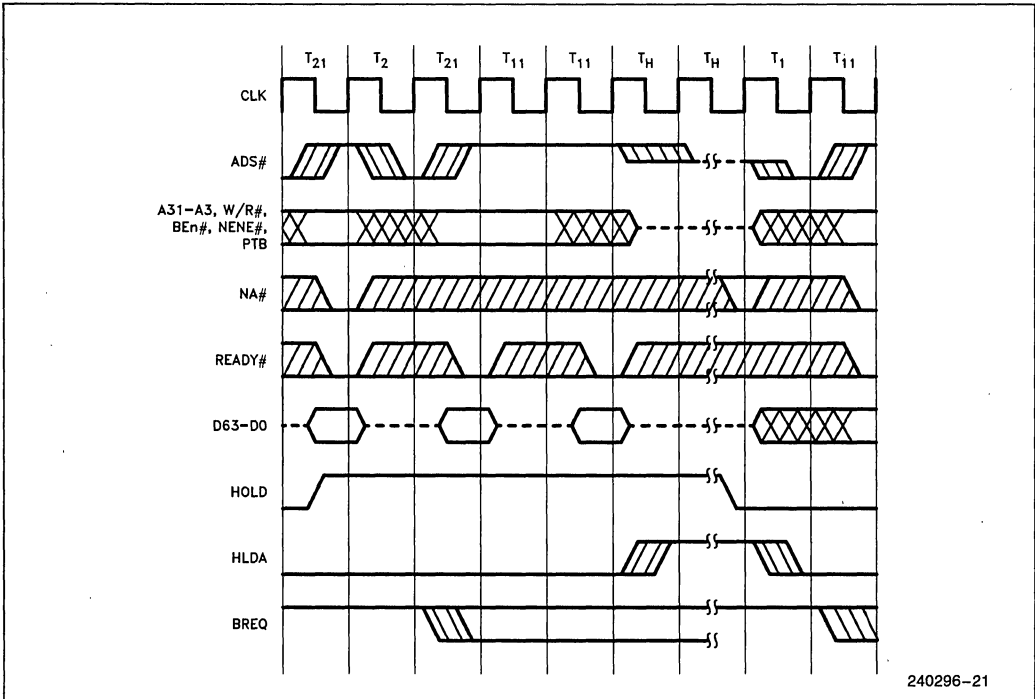


Figure 4.10. HOLD, HLDA, and BREQ

If, during a HOLD cycle, an internal bus request is generated, BREQ is activated even though HLDA is asserted. It remains active at least until the clock after ADS# is activated for the requested cycle.

SET. If INT/CS8 is sampled active, the i860 microprocessor enters CS8 mode. No inputs (except for HOLD, INT/CS8, and CC1-CC0) are sampled during RESET.

4.4 Bus States During RESET

Figure 4.11 shows how INT/CS8 is sampled during the clock period just before the falling edge of RE-

Note that, because HOLD is recognized even while RESET is active, the HLDA output signal may also become active during RESET. Refer to Table 3.4 "Output Pin Status during Reset".

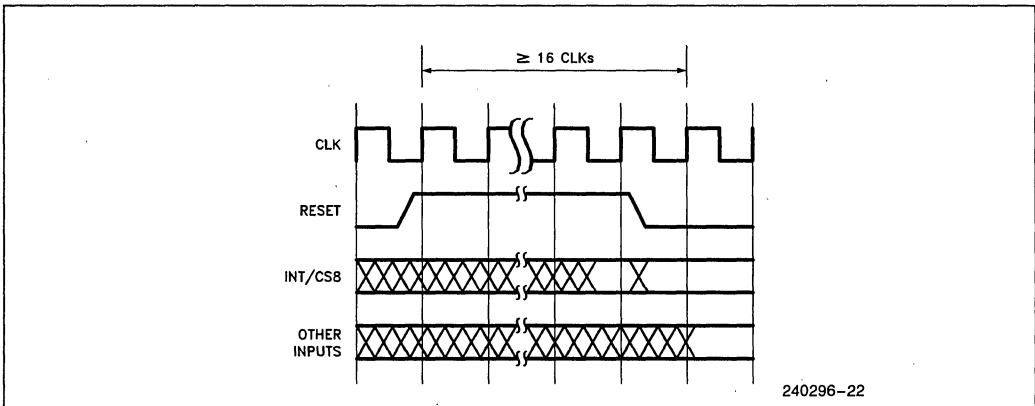


Figure 4.11. Reset Activities

5.0 MECHANICAL DATA

Figures 5.1 and 5.2 show the locations of pins; Tables 5.1 and 5.2 help to locate pin identifiers.

	S	R	Q	P	N	M	L	K	J	H	G	F	E	D	C	B	A	
1	() V _{CC}	() V _{SS}	() V _{CC}	() V _{SS}	() A12	() A17	() A19	() A21	() A23	() A25	() A29	() A31	() V _{CC}	() V _{SS}	() V _{CC}	() V _{SS}	() V _{CC}	1
2	() V _{SS}	() V _{CC}	() V _{SS}	() A8	() A10	() A13	() A15	() A18	() A20	() A24	() A27	() A28	() CC0	() V _{CC}	() V _{SS}	() V _{CC}	() V _{SS}	2
3	() V _{CC}	() V _{SS}	() A6	() A7	() A9	() A11	() A14	() A16	() CLK	() A22	() A26	() A30	() CC1	() D62	() D60	() V _{SS}	() V _{CC}	3
4	() V _{SS}	() V _{CC}	() A5												() D63	() D59	() V _{SS}	4
5	() V _{CC}	() A4	() A3												() D61	() D58	() D56	5
6	() W/R#	() NENE#	() PTB												() D57	() D54	() D52	6
7	() ADS#	() HLDA	() BREQ												() D55	() D53	() D50	7
8	() LOCK#	() KEN#	() READY#												() D51	() D49	() D48	8
9	() INT/CSB	() NA#	() HOLD												() D47	() D45	() D46	9
10	() BE5#	() BE7#	() BE6#												() D43	() D42	() D44	10
11	() BE3#	() BE2#	() BE4#												() D39	() D41	() D40	11
12	() SHI	() BE1#	() BE0#												() D37	() D36	() D38	12
13	() RESET	() SCAN	() BSCN												() D35	() D34	() V _{CC}	13
14	() V _{SS}	() D0	() D1												() D33	() V _{CC}	() V _{SS}	14
15	() V _{CC}	() V _{SS}	() D2	() D3	() D5	() D7	() D11	() D13	() D17	() D21	() D23	() D27	() D29	() D31	() D32	() V _{SS}	() V _{CC}	15
16	() V _{SS}	() V _{CC}	() V _{SS}	() V _{CC}	() D4	() D9	() D8	() D15	() D14	() D19	() D22	() D25	() D28	() D30	() V _{SS}	() V _{CC}	() V _{SS}	16
17	() V _{CC}	() V _{SS}	() V _{CC}	() V _{SS}	() V _{CC}	() D6	() D10	() D12	() D16	() D18	() D20	() D24	() D26	() V _{SS}	() V _{CC}	() V _{SS}	() V _{CC}	17

240296-23

Figure 5.1. Pin Configuration—View from Top Side

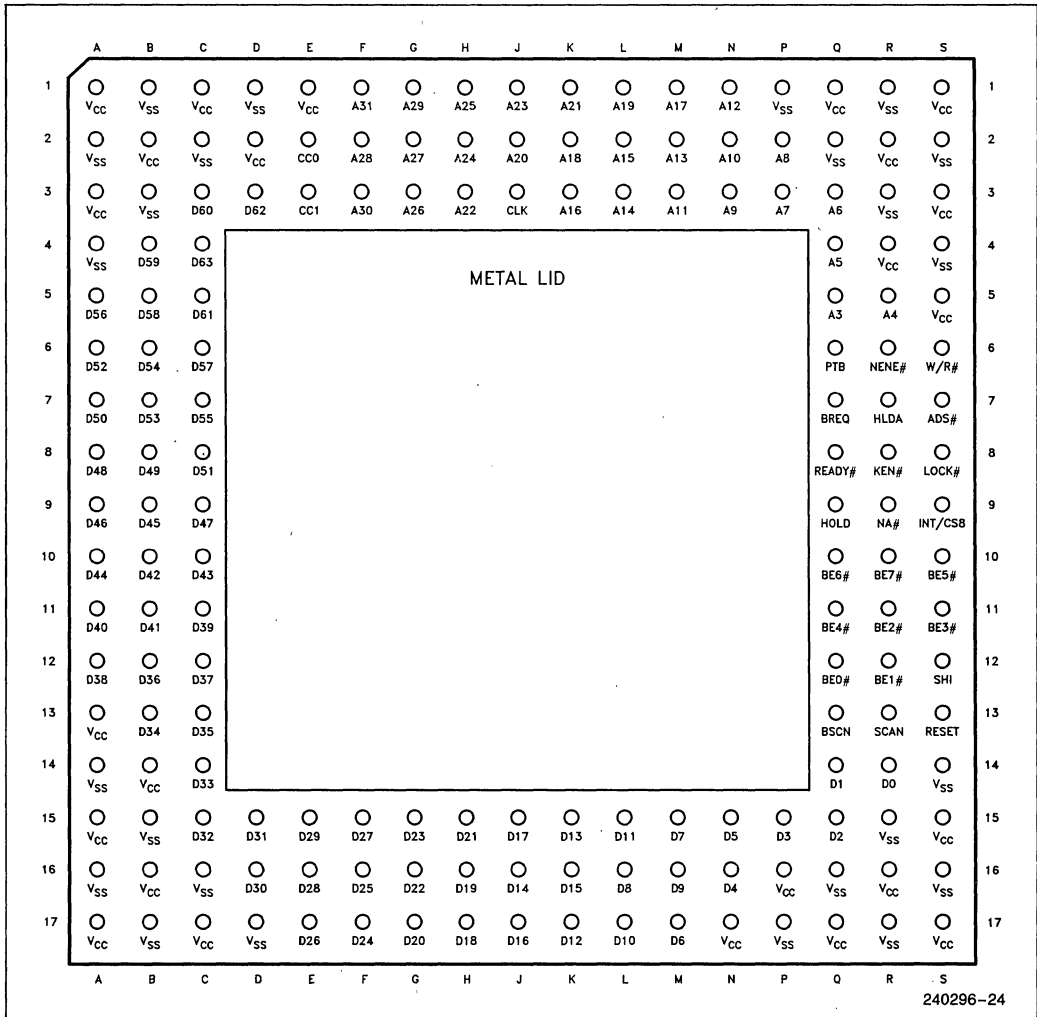


Figure 5.2. Pin Configuration—View from Pin Side

Table 5.1. Pin Cross Reference by Location

Location	Signal	Location	Signal	Location	Signal	Location	Signal
A1	VCC	C9	D47	J15	D17	Q10	BE6#
A2	VSS	C10	D43	J16	D14	Q11	BE4#
A3	VCC	C11	D39	J17	D16	Q12	BE0#
A4	VSS	C12	D37	K1	A21	Q13	BSCN
A5	D56	C13	D35	K2	A18	Q14	D1
A6	D52	C14	D33	K3	A16	Q15	D2
A7	D50	C15	D32	K15	D13	Q16	VSS
A8	D48	C16	VSS	K16	D15	Q17	VCC
A9	D46	C17	VCC	K17	D12	R1	VSS
A10	D44	D1	VSS	L1	A19	R2	VCC
A11	D40	D2	VCC	L2	A15	R3	VSS
A12	D38	D3	D62	L3	A14	R4	VCC
A13	VCC	D15	D31	L15	D11	R5	A4
A14	VSS	D16	D30	L16	D8	R6	NENE#
A15	VCC	D17	VSS	L17	D10	R7	HLDA
A16	VSS	E1	VCC	M1	A17	R8	KEN#
A17	VCC	E2	CC0	M2	A13	R9	NA#
B1	VSS	E3	CC1	M3	A11	R10	BE7#
B2	VCC	E15	D29	M15	D7	R11	BE2#
B3	VSS	E16	D28	M16	D9	R12	BE1#
B4	D59	E17	D26	M17	D6	R13	SCAN
B5	D58	F1	A31	N1	A12	R14	D0
B6	D54	F2	A28	N2	A10	R15	VSS
B7	D53	F3	A30	N3	A9	R16	VCC
B8	D49	F15	D27	N15	D5	R17	VSS
B9	D45	F16	D25	N16	D4	S1	VCC
B10	D42	F17	D24	N17	VCC	S2	VSS
B11	D41	G1	A29	P1	VSS	S3	VCC
B12	D36	G2	A27	P2	A8	S4	VSS
B13	D34	G3	A26	P3	A7	S5	VCC
B14	VCC	G15	D23	P15	D3	S6	W/R#
B15	VSS	G16	D22	P16	VCC	S7	ADS#
B16	VCC	G17	D20	P17	VSS	S8	LOCK#
B17	VSS	H1	A25	Q1	VCC	S9	INT/CS8
C1	VCC	H2	A24	Q2	VSS	S10	BE5#
C2	VSS	H3	A22	Q3	A6	S11	BE3#
C3	D60	H15	D21	Q4	A5	S12	SHI
C4	D63	H16	D19	Q5	A3	S13	RESET
C5	D61	H17	D18	Q6	PTB	S14	VSS
C6	D57	J1	A23	Q7	BREQ	S15	VCC
C7	D55	J2	A20	Q8	READY#	S16	VSS
C8	D51	J3	CLK	Q9	HOLD	S17	VCC

Table 5.2. Pin Cross Reference by Pin Name

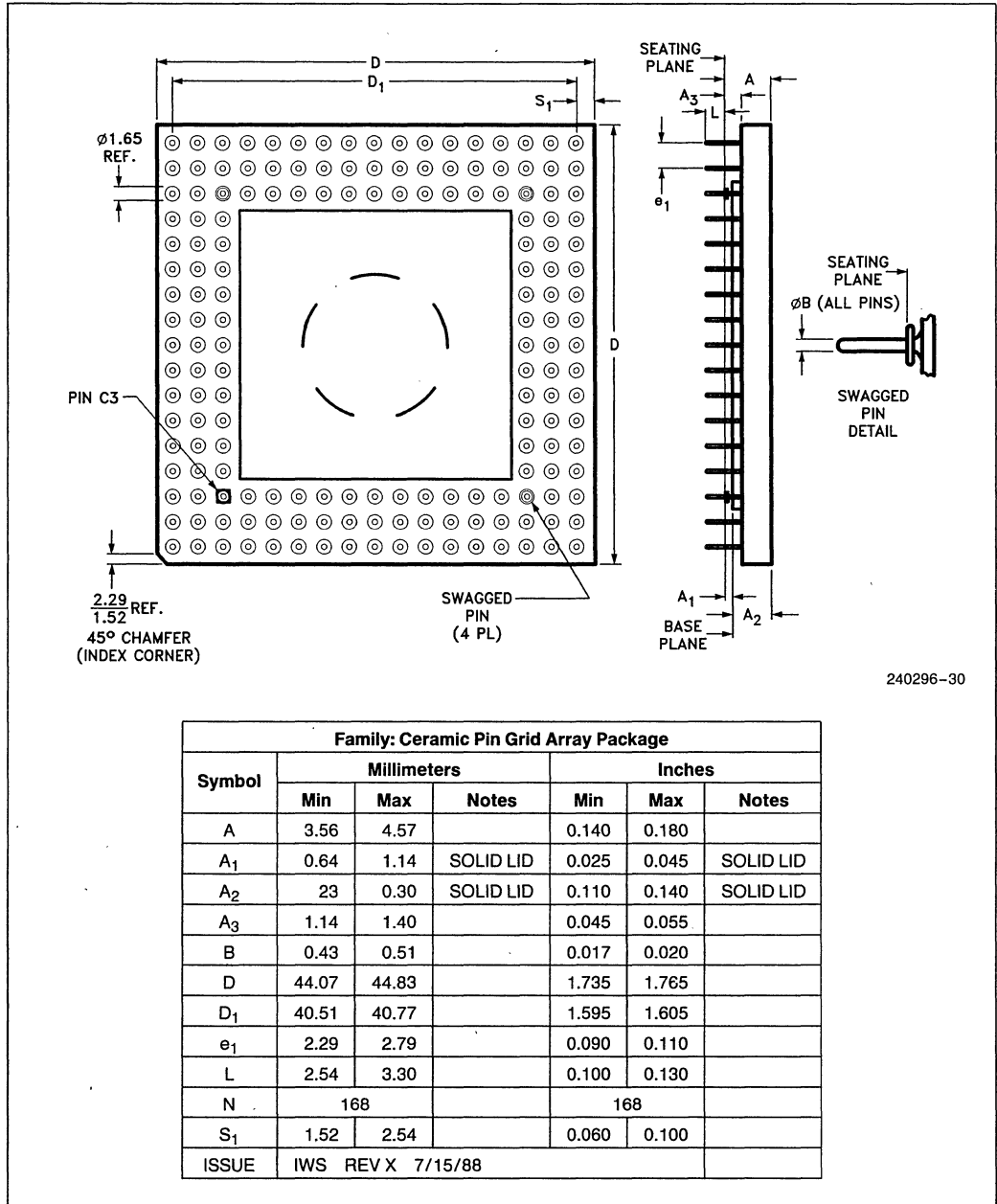
Signal	Location	Signal	Location	Signal	Location	Signal	Location
A3	Q5	CLK	J3	D41	B11	VCC	B16
A4	R5	D0	R14	D42	B10	VCC	C1
A5	Q4	D1	Q14	D43	C10	VCC	C17
A6	Q3	D2	Q15	D44	A10	VCC	D2
A7	P3	D3	P15	D45	B9	VCC	E1
A8	P2	D4	N16	D46	A9	VCC	N17
A9	N3	D5	N15	D47	C9	VCC	P16
A10	N2	D6	M17	D48	A8	VCC	Q1
A11	M3	D7	M15	D49	B8	VCC	Q17
A12	N1	D8	L16	D50	A7	VCC	R2
A13	M2	D9	M16	D51	C8	VCC	R4
A14	L3	D10	L17	D52	A6	VCC	R16
A15	L2	D11	L15	D53	B7	VCC	S1
A16	K3	D12	K17	D54	B6	VCC	S3
A17	M1	D13	K15	D55	C7	VCC	S5
A18	K2	D14	J16	D56	A5	VCC	S15
A19	L1	D15	K16	D57	C6	VCC	S17
A20	J2	D16	J17	D58	B5	VSS	A2
A21	K1	D17	J15	D59	B4	VSS	A4
A22	H3	D18	H17	D60	C3	VSS	A14
A23	J1	D19	H16	D61	C5	VSS	A16
A24	H2	D20	G17	D62	D3	VSS	B1
A25	H1	D21	H15	D63	C4	VSS	B3
A26	G3	D22	G16	HLDA	R7	VSS	B15
A27	G2	D23	G15	HOLD	Q9	VSS	B17
A28	F2	D24	F17	INT/CS8	S9	VSS	C2
A29	G1	D25	F16	KEN#	R8	VSS	C16
A30	F3	D26	E17	LOCK#	S8	VSS	D1
A31	F1	D27	F15	NA#	R9	VSS	D17
ADS#	S7	D28	E16	NENE#	R6	VSS	P1
BE0#	Q12	D29	E15	PTB	Q6	VSS	P17
BE1#	R12	D30	D16	READY#	Q8	VSS	Q2
BE2#	R11	D31	D15	RESET	S13	VSS	Q16
BE3#	S11	D32	C15	SCAN	R13	VSS	R1
BE4#	Q11	D33	C14	SHI	S12	VSS	R3
BE5#	S10	D34	B13	VCC	A1	VSS	R15
BE6#	Q10	D35	C13	VCC	A3	VSS	R17
BE7#	R10	D36	B12	VCC	A13	VSS	S2
BREQ	Q7	D37	C12	VCC	A15	VSS	S4
BSCN	Q13	D38	A12	VCC	A17	VSS	S14
CC0	E2	D39	C11	VCC	B2	VSS	S16
CC1	E3	D40	A11	VCC	B14	W/R#	S6

Table 5.3. Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A ₁	Distance between seating plane and base plane (lid)
A ₂	Distance from base plane to highest point of body
A ₃	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D ₁	A body length dimension, outer lead center to outer lead center
e ₁	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S ₁	Other body dimension, outer lead center to edge of body

NOTES:

1. Controlling dimension: millimeter.
2. Dimension "e₁" ("e") is non-cumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B₁" and "C" are nominal.
5. Details of Pin 1 identifier are optional.



240296-30

Figure 5.3. 168 Lead Ceramic PGA Package Dimensions

6.0 PACKAGE THERMAL SPECIFICATIONS

The i860 microprocessor is specified for operation when T_C (the case temperature) is within the range of 0°C–85°C. T_C may be measured in any environment to determine whether the i860 microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

T_A (the ambient temperature) can be calculated from θ_{CA} (thermal resistance from case to ambient) with the following equation:

$$T_A = T_C - P \cdot \theta_{CA}$$

Typical values for θ_{CA} at various airflows are given in Table 6.1 for the 1.75 sq. in., 168 pin, ceramic PGA.

Table 6.2 shows the maximum T_A allowable (without exceeding T_C) at various airflows and operating frequencies (f_{CLK}).

Note that T_A is greatly improved by attaching "fins" or a "heat sink" to the package. P (the maximum power consumption) is calculated by using the maximum I_{CC} at 5V as tabulated in the *DC Characteristics* of section 7.

Table 6.1. Thermal Resistance (θ_{CA}) at Various Airflows

	In °C/Watt					
	Airflow-ft/min (m/sec)					
	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
θ_{CA} with Heat Sink*	13	9	5.5	5.0	3.9	3.4
θ_{CA} without Heat Sink	17	14	11	9	7.1	6.6

*0.285" high unidirectional heat sink (Al alloy 6061, 50 mil fin width, 150 mil center-to-center fin spacing).

Table 6.2. Maximum T_A at Various Airflows

	f_{CLK} (MHz)	In °C					
		Airflow-ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T_A with Heat Sink*	33.3	46	58	69	70	73	75
	40.0	43	56	67	69	72	74
T_A without Heat Sink	33.3	34	43	52	58	64	65
	40.0	30	40	49	56	62	64

*0.285" high unidirectional heat sink (Al alloy 6061, 50 mil fin width, 150 mil center-to-center fin spacing).

7.0 ELECTRICAL DATA

Inputs and outputs are TTL compatible. All input and output timings are specified relative to the 1.5 volt level of the rising edge of CLK and refer to the point that the signals reach 1.5V.

7.1 Absolute Maximum Ratings

Case Temperature T_C under Bias 0°C to 85°C
 Storage Temperature -65°C to +150°C
 Voltage on Any Pin
 with Respect to Ground -0.5 to 6.5V

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

7.2 D.C. Characteristics

Table 7.1. DC Characteristics
 $T_C = 0^\circ\text{C to } 85^\circ\text{C}, V_{CC} = 5\text{V} \pm 5\%$

Symbol	Parameter	Min	Max	Units	Notes
V_{IL}	Input LOW Voltage	-0.3	+0.8	V	
V_{IH}	Input HIGH Voltage	2.0	$V_{CC} + 0.3$	V	
V_{ILC}	CLK Input LOW Voltage	-0.3	+0.8	V	
V_{IHC}	CLK Input HIGH Voltage	3.0	$V_{CC} + 0.3$	V	
V_{OL}	Output LOW Voltage		0.45	V	(Note 1)
V_{OH}	Output HIGH Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current				
	CLK = 33.3 MHz		600	mA	$V_{CC} @ 5\text{V}$
	CLK = 40.0 MHz		650	mA	$V_{CC} @ 5\text{V}$
I_{LI}	Input Leakage Current		± 15	μA	No pullup or pulldown
I_{LO}	Output Leakage Current		± 15	μA	
C_{IN}	Input Capacitance		15	pF	(Note 3)
C_O	I/O or Output Capacitance		15	pF	(Note 3)
C_{CLK}	Clock Capacitance		20	pF	(Note 3)

NOTES:

1. This parameter is measured at 4.0 mA for A31-A3, D63-D0, BE7#-BE0#; at 5.0 mA for all other outputs.
2. This parameter is measured at 1.0 mA for A31-A3, D63-D0, BE7#-BE0#; at 0.9 mA all other outputs.
3. These are not tested. They are guaranteed by design characterization.

7.3 A.C. Characteristics

Table 7.2. A.C. Characteristics

T_C = 0°C to 85°C, V_{CC} = 5V ± 5%

All timings measured at CLK = 1.5V unless otherwise specified.

Symbol	Parameter	33.3 MHz		40.0 MHz		Test Conditions
		Min (ns)	Max (ns)	Min (ns)	Max (ns)	
t1	CLK period	30	125	25	125	
t2	CLK high time	7		5		at 3V
t3	CLK low time	7		5		at 0.8V
t4	CLK fall time		4		4	2V to 0.8V
t5	CLK rise time		4		4	0.8V to 2V
t6a	A31–A3, PTB, W/R#, NENE# valid delay	3.5	23	3.5	19	50 pF load
t6b	BE _n #* valid delay	3.5	25	3.5	21	50 pF load
t7	Float time, all outputs	3.5	30	3.5	25	(Note 1)
t8	ADS#, BREQ, LOCK#, HLDA valid delay	3.5	20	3.5	15	50 pF load
t9	D63–D0 valid delay	3.5	35	3.5	31	50 pF load
t10	Setup time, all inputs except INT, HOLD	11		8		
t11	Hold time, all inputs except INT, HOLD	4		3		
t12	INT, HOLD setup time	11		8		(Note 2)
t13	INT, HOLD hold time	4		4		(Note 2)

NOTES:

1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not tested.
2. INT and HOLD are asynchronous inputs. The setup and hold specifications are given for test purposes or to assure recognition on a specific rising edge of CLK.

*n = 0, 1, ..., 7

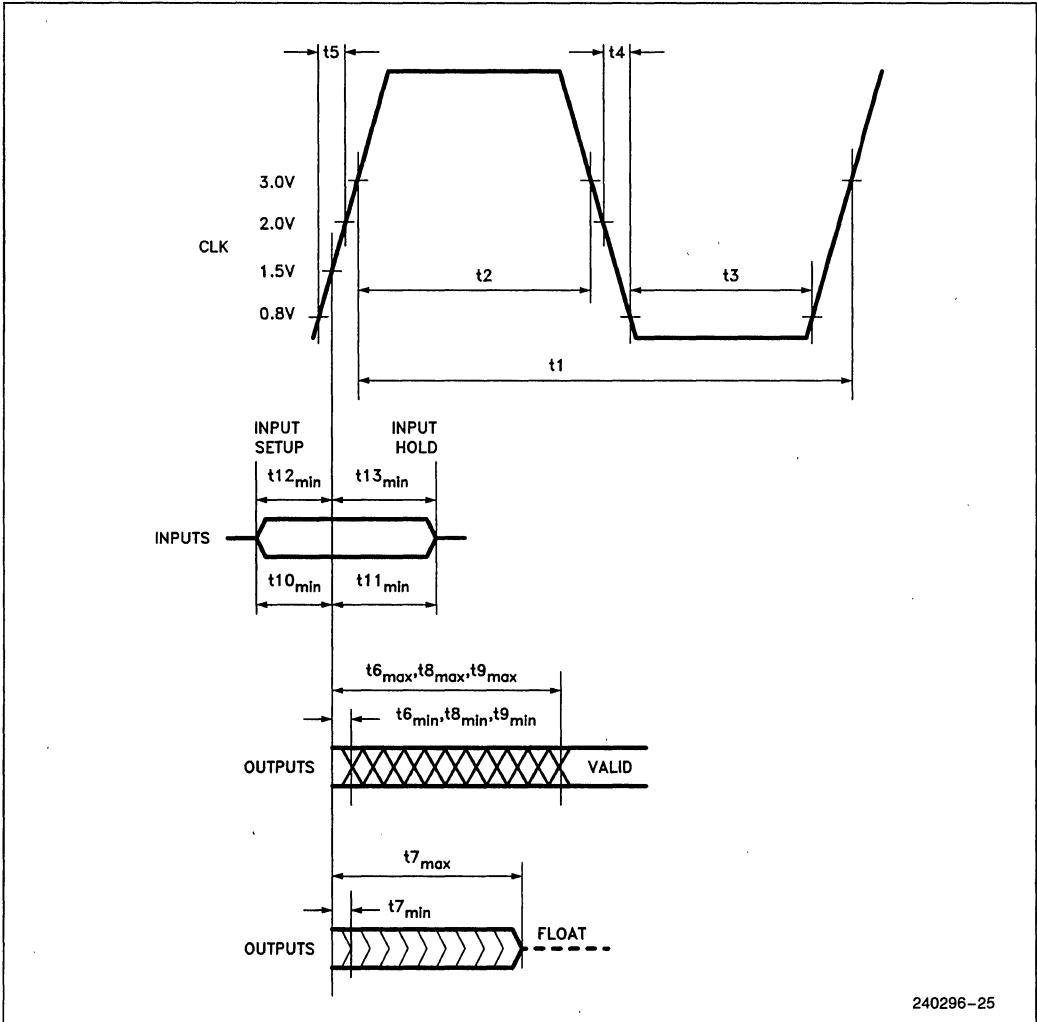


Figure 7.1. CLK, Input, and Output Timings

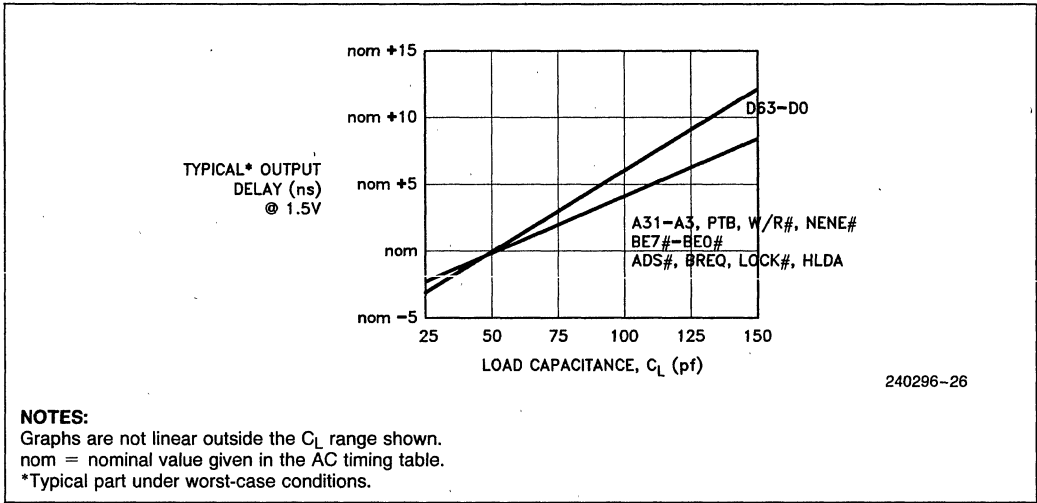


Figure 7.2. Typical Output Delay vs Load Capacitance under Worst-Case Conditions

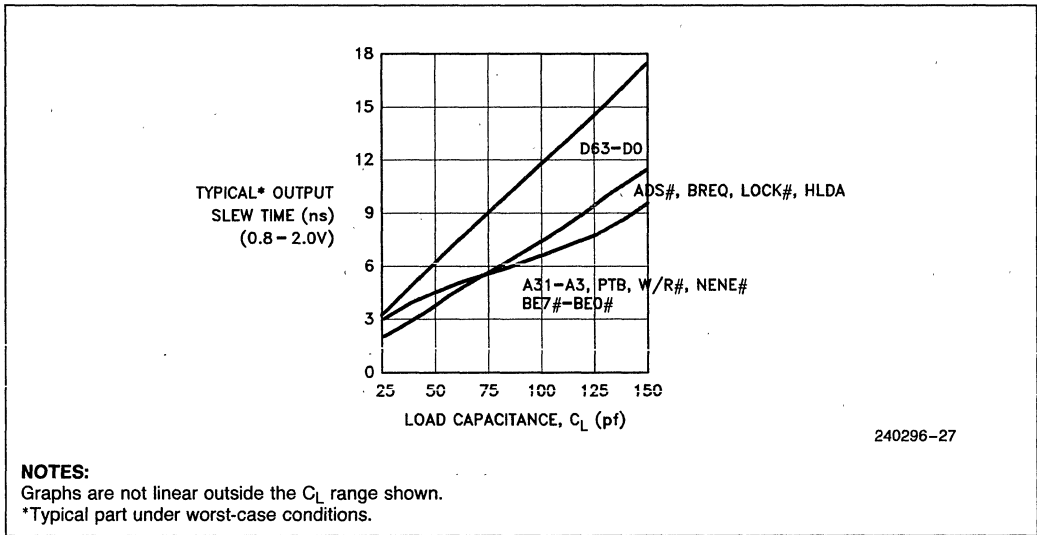


Figure 7.3. Typical Slew Time vs Load Capacitance under Worst-Case Conditions

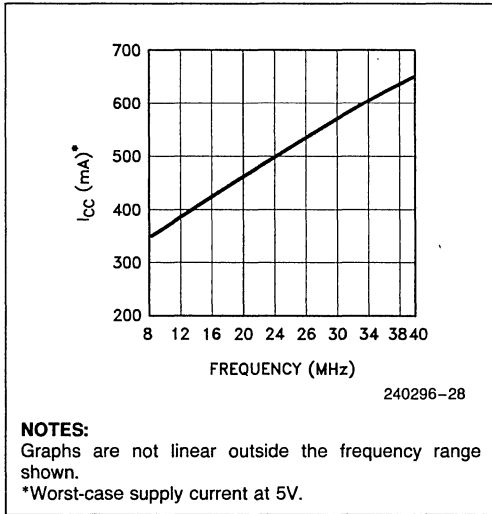


Figure 7.4. Typical ICC vs Frequency

8.0 INSTRUCTION SET

Key to abbreviations:

- src1* A register (integer or floating-point depending on class of instruction) or a 16-bit immediate value. The immediate value is sign-extended for add and subtract operations and zero-extended for logical operations.
- src1ni* Same as *src1* except that no immediate value is permitted.
- src2* A register (integer or floating-point depending on class of instruction).
- rdest* A register (integer or floating-point depending on class of instruction).
- freg* A floating-point register.
- ireg* An integer register.
- ctrlreg* One of the control registers *fir*, *psr*, *epsr*, *dirbase*, *db*, or *fsr*.
- #const* A 16-bit immediate address offset that the i860 microprocessor sign-extends to 32 bits when computing the effective address.

mem.x(address) The contents of the memory location indicated by *address* with a size of *x*.

Table 8.1. Precision Specification

Suffix	Source Precision	Result Precision
.ss	single	single
.sd	single	double
.dd	double	double
.ds	double	single

Unless otherwise specified, floating-point operations accept single- or double-precision source operands and produce a result of equal or greater precision. Both input operands must have the same precision. The source and result precision are specified by a two-letter suffix to the mnemonic of the operation.

- .p* Precision specification *.ss*, *.sd*, or *.dd* (*.ds* not permitted). Refer to Table 8.1.
- .r* Precision specification *.ss*, *.sd*, *.ds*, or *.dd*. Refer to Table 8-1.
- .w* *.ss* (32 bits), or *.dd* (64 bits)
- .x* *.b* (8 bits), *.s* (16 bits), or *.l* (32 bits)
- .y* *.l* (32 bits), *.d* (64 bits), or *.q* (128 bits)
- .z* *.l* (32 bits), or *.d* (64 bits)
- lbroff* A signed, 26-bit, immediate, relative branch offset
- sbroff* A signed, 16-bit, immediate, relative branch offset
- brx* A function that computes the target address of a branch by shifting the offset (either *lbroff* or *sbroff*) left by two bits, sign-extending it to 32 bits, and adding the result to the address of the current control-transfer instruction plus four.
- src1s* An integer register or a 5-bit immediate that is zero-extended to 32 bits.
- PM The pixel mask, which is considered as an array of eight bits PM[0]..PM[7], where PM[0] is the least significant bit.

8.1 Instruction Definitions in Alphabetical Order

adds	<i>src1, src2, rdest</i>	Add Signed
	<i>rdest</i> ← <i>src1</i> + <i>src2</i>	
	OF ← (bit 31 carry ≠ bit 30 carry)	
	CC set if <i>src2</i> < - <i>src1</i> (signed)	
	CC clear if <i>src2</i> ≥ - <i>src1</i> (signed)	
addu	<i>src1, src2, rdest</i>	Add Unsigned
	<i>rdest</i> ← <i>src1</i> + <i>src2</i>	
	OF ← bit 31 carry	
	CC ← bit 31 carry	
and	<i>src1, src2, rdest</i>	Logical AND
	<i>rdest</i> ← <i>src1</i> and <i>src2</i>	
	CC set if result is zero, cleared otherwise	
andh	<i>#const, src2, rdest</i>	Logical AND High
	<i>rdest</i> ← (<i>#const</i> shifted left 16 bits) and <i>src2</i>	
	CC set if result is zero, cleared otherwise	
andnot	<i>src1, src2, rdest</i>	Logical AND NOT
	<i>rdest</i> ← not <i>src1</i> and <i>src2</i>	
	CC set if result is zero, cleared otherwise	
andnoth	<i>#const, src2, rdest</i>	Logical AND NOT High
	<i>rdest</i> ← not (<i>#const</i> shifted left 16 bits) and <i>src2</i>	
	CC set if result is zero, cleared otherwise	
bc	<i>lbroff</i>	Branch on CC
	IF CC = 1	
	THEN continue execution at <i>brx(lbroff)</i>	
	FI	
bc.t	<i>lbroff</i>	Branch on CC, Taken
	IF CC = 1	
	THEN execute one more sequential instruction	
	continue execution at <i>brx(lbroff)</i>	
	ELSE skip next sequential instruction	
	FI	
bla	<i>src1ni, src2, sbroff</i>	Branch on LCC and Add
	LCC-temp clear if <i>src2</i> < - <i>src1ni</i> (signed)	
	LCC-temp set if <i>src2</i> ≥ - <i>src1ni</i> (signed)	
	<i>src2</i> ← <i>src1ni</i> + <i>src2</i>	
	Execute one more sequential instruction	
	IF LCC	
	THEN LCC ← LCC-temp	
	continue execution at <i>brx(sbroff)</i>	
	ELSE LCC ← LCC-temp	
	FI	
bnc	<i>lbroff</i>	Branch on Not CC
	IF CC = 0	
	THEN continue execution at <i>brx(lbroff)</i>	
	FI	
bnc.t	<i>lbroff</i>	Branch on Not CC, Taken
	IF CC = 0	
	THEN execute one more sequential instruction	
	continue execution at <i>brx(lbroff)</i>	
	ELSE skip next sequential instruction	
	FI	
br	<i>lbroff</i>	Branch Direct Unconditionally
	Execute one more sequential instruction.	
	Continue execution at <i>brx(lbroff)</i> .	

- bri** *[src1ni]* **Branch Indirect Unconditionally**
 Execute one more sequential instruction
 IF any trap bit in **psr** is set
 THEN copy PU to U, PIM to IM in **psr**
 clear trap bits
 IF DS is set and DIM is reset
 THEN enter dual-instruction mode after executing one
 instruction in single-instruction mode
 ELSE IF DS is set and DIM is set
 THEN enter single-instruction mode after executing one
 instruction in dual-instruction mode
 ELSE IF DIM is set
 THEN enter dual-instruction mode
 for next two instructions
 ELSE enter single-instruction mode
 for next two instructions
 FI
 FI
 FI
 Continue execution at address in *src1ni*
 (The original contents of *src1ni* is used even if the next instruction
 modifies *src1ni*. Does not trap if *src1ni* is misaligned.)
- bte** *src1s, src2, sbroff* **Branch If Equal**
 IF *src1s = src2*
 THEN continue execution at *brx(sbroff)*
 FI
- btne** *src1s, src2, sbroff* **Branch If Not Equal**
 IF *src1s ≠ src2*
 THEN continue execution at *brx(sbroff)*
 FI
- call** *lbroff* **Subroutine Call**
 r1 ← address of next sequential instruction + 4
 Execute one more sequential instruction
 Continue execution at *brx(lbroff)*
- calli** *[src1ni]* **Indirect Subroutine Call**
 r1 ← address of next sequential instruction + 4
 Execute one more sequential instruction
 Continue execution at address in *src1ni*
 (The original contents of *src1ni* is used even if the next instruction
 modifies *src1ni*. Does not trap if *src1ni* is misaligned.
 The register *src1ni* must not be r1.)
- fadd.p** *src1, src2, rdest* **Floating-Point Add**
rdest ← *src1 + src2*
- faddp** *src1, src2, rdest* **Add with Pixel Merge**
rdest ← *src1 + src2*
 Shift and load MERGE register as defined in Table 8.2
- faddz** *src1, src2, rdest* **Add with Z Merge**
rdest ← *src1 + src2*
 Shift MERGE right 16 and load fields 31..16 and 63..48
- famov.r** *src1, rdest* **Floating-Point Adder Move**
rdest ← *src1*
 Send *src1* through the floating-point adder. (Preserves -0 (minus zero) when *src1* is -0. *src2*
 must be coded as f0 by the assembler.)
- fiadd.w** *src1, src2, rdest* **Long-Integer Add**
rdest ← *src1 + src2*

fisub.w	<i>src1, src2, rdest</i>	Long-Integer Subtract
	<i>rdest</i> ← <i>src1</i> - <i>src2</i>	
fix.p	<i>src1, rdest</i>	Floating-Point to Integer Conversion
	<i>rdest</i> ← 64-bit value with low-order 32 bits equal to integer part of <i>src1</i> rounded	
		Floating-Point Load
fld.y	<i>src1(src2), freg</i>	(Normal)
fld.y	<i>src1(src2)++</i> , <i>freg</i>	(Autoincrement)
	<i>freg</i> ← mem.y (<i>src1</i> + <i>src2</i>)	
	IF autoincrement	
	THEN <i>src2</i> ← <i>src1</i> + <i>src2</i>	
	FI	
		Cache Flush
flush	# <i>const(src2)</i>	(Normal)
flush	# <i>const(src2)++</i>	(Autoincrement)
	Replace block in data cache with address (# <i>const</i> + <i>src2</i>).	
	Contents of block undefined.	
	IF autoincrement	
	THEN <i>src2</i> ← # <i>const</i> + <i>src2</i>	
	FI	
fmlow.dd	<i>src1, src2, rdest</i>	Floating-Point Multiply Low
	<i>rdest</i> ← low-order 53 bits of <i>src1</i> mantissa × <i>src2</i> mantissa	
	<i>rdest</i> bit 53 ← most significant bit of mantissa	
fmov.r	<i>src1, rdest</i>	Floating-Point Reg-Reg Move
	Assembler pseudo-operation	
	fmov.ss <i>src1, rdest</i> = fiadd.ss <i>src1, f0, rdest</i>	
	fmov.dd <i>src1, rdest</i> = fiadd.dd <i>src1, f0, rdest</i>	
	fmov.sd <i>src1, rdest</i> = famov.sd <i>src1, rdest</i>	
	fmov.ds <i>src1, rdest</i> = famov.ds <i>src1, rdest</i>	
fmul.p	<i>src1, src2, rdest</i>	Floating-Point Multiply
	<i>rdest</i> ← <i>src1</i> × <i>src2</i>	
fnop	Floating-Point No Operation
	Assembler pseudo-operation	
	fnop = shrd <i>r0, r0, r0</i>	
form	<i>src1, rdest</i>	OR with MERGE Register
	<i>rdest</i> ← <i>src1</i> OR MERGE	
	MERGE ← 0	
frcp.p	<i>src2, rdest</i>	Floating-Point Reciprocal
	<i>rdest</i> ← 1/ <i>src2</i> with maximum mantissa error < 2 ⁻⁷	
frsqr.p	<i>src2, rdest</i>	Floating-Point Reciprocal Square Root
	<i>rdest</i> ← 1/SQRT (<i>src2</i>) with maximum mantissa error < 2 ⁻⁷	
		Floating-Point Store
fst.y	<i>freg, src1(src2)</i>	(Normal)
fst.y	<i>freg, src1(src2)++</i>	(Autoincrement)
	mem.y (<i>src2</i> + <i>src1</i>) ← <i>freg</i>	
	IF autoincrement	
	THEN <i>src2</i> ← <i>src1</i> + <i>src2</i>	
	FI	
fsub.p	<i>src1, src2, rdest</i>	Floating-Point Subtract
	<i>rdest</i> ← <i>src1</i> - <i>src2</i>	
fttrunc.p	<i>src1, rdest</i>	Floating-Point to Integer Conversion
	<i>rdest</i> ← 64-bit value with low-order 32 bits equal to integer part of <i>src1</i>	
fxfr	<i>src1, ireg</i>	Transfer F-P to Integer Register
	<i>ireg</i> ← <i>src1</i>	

fzchk1	<i>src1, src2, rdest</i>	32-Bit Z-Buffer Check
	Consider <i>src1</i> , <i>src2</i> , and <i>rdest</i> as arrays of two 32-bit fields <i>src1</i> (0).. <i>src1</i> (1), <i>src2</i> (0).. <i>src2</i> (1), and <i>rdest</i> (0).. <i>rdest</i> (1) where zero denotes the least-significant field.	
	PM ← PM shifted right by 2 bits	
	FOR i = 0 to 1	
	DO	
	PM [i + 6] ← <i>src2</i> (i) ≤ <i>src1</i> (i) (unsigned)	
	<i>rdest</i> (i) ← smaller of <i>src2</i> (i) and <i>src1</i> (i)	
	OD	
	MERGE ← 0	
fzchks	<i>src1, src2, rdest</i>	16-Bit Z-Buffer Check
	Consider <i>src1</i> , <i>src2</i> , and <i>rdest</i> as arrays of four 16-bit fields <i>src1</i> (0).. <i>src1</i> (3), <i>src2</i> (0).. <i>src2</i> (3), and <i>rdest</i> (0).. <i>rdest</i> (3) where zero denotes the least-significant field.	
	PM ← PM shifted right by 4 bits	
	FOR i = 0 to 3	
	DO	
	PM [i + 4] ← <i>src2</i> (i) ≤ <i>src1</i> (i) (unsigned)	
	<i>rdest</i> (i) ← smaller of <i>src2</i> (i) and <i>src1</i> (i)	
	OD	
	MERGE ← 0	
intovr	Software Trap on Integer Overflow
	If OF in epsr = 1, generate trap with IT set in psr .	
ixfr	<i>src1ni, freg</i>	Transfer Integer to F-P Register
	<i>freg</i> ← <i>src1ni</i>	
ld.c	<i>ctr1reg, rdest</i>	Load from Control Register
	<i>rdest</i> ← <i>ctr1reg</i>	
ld.x	<i>src1(src2), rdest</i>	Load Integer
	<i>rdest</i> ← <i>mem.x</i> (<i>src1</i> + <i>src2</i>)	
lock	Begin Interlocked Sequence
	Set BL in dirbase . The next load or store locks the bus.	
	Disable interrupts until the bus is unlocked.	
mov	<i>src2, rdest</i>	Register-Register Move
	Assembler pseudo-operation	
	mov <i>src2, rdest</i> = shl <i>r0, src2, rdest</i>	
nop	Core-Unit No Operation
	Assembler pseudo-operation	
	nop = shl <i>r0, r0, r0</i>	
or	<i>src1, src2, rdest</i>	Logical OR
	<i>rdest</i> ← <i>src1</i> OR <i>src2</i>	
	CC set if result is zero, cleared otherwise	
orh	<i>#const, src2, rdest</i>	Logical OR High
	<i>rdest</i> ← (<i>#const</i> shifted left 16 bits) OR <i>src2</i>	
	CC set if result is zero, cleared otherwise	
pfadd.p	<i>src1, src2, rdest</i>	Pipelined Floating-Point Add
	<i>rdest</i> ← last stage Adder result	
	Advance A pipeline one stage	
	A pipeline first stage ← <i>src1</i> + <i>src2</i>	
pfaddp	<i>src1, src2, rdest</i>	Pipelined Add with Pixel Merge
	<i>rdest</i> ← last stage Graphics result	
	last stage Graphics result ← <i>src1</i> + <i>src2</i>	
	Shift and load MERGE register from last stage Graphics result as defined in Table 8.2	

- pfaddz** *src1, src2, rdest* Pipelined Add with Z Merge
rdest ← last stage Graphics result
 last stage Graphics result ← *src1* + *src2*
 Shift MERGE right 16 and load fields 31..16 and 63..48 from last stage Graphics result
- pfam.p** *src1, src2, rdest* Pipelined Floating-Point Add and Multiply
rdest ← last stage Adder result
 Advance A and M pipeline one stage (operands accessed before advancing pipeline)
 A pipeline first stage ← A-op1 + A-op2
 M pipeline first stage ← M-op1 × M-op2
- pfamov.r** *src1, rdest* Pipelined Floating-Point Adder Move
rdest ← last stage Adder result
 Advance A pipeline one stage
 A pipeline first stage ← *src1*
- pfreq.p** *src1, src2, rdest* Pipelined Floating-Point Equal Compare
rdest ← last stage Adder result
 CC set if *src1* = *src2*, else cleared
 Advance A pipeline one stage
 A pipeline first stage is undefined, but no result exception occurs
- pfgt.p** *src1, src2, rdest* Pipelined Floating-Point Greater-Than Compare
 (Assembler clears R-bit of instruction)
rdest ← last stage Adder result
 CC set if *src1* > *src2*, else cleared
 Advance A pipeline one stage
 A pipeline first stage is undefined, but no result exception occurs
- pfiaadd.w** *src1, src2, rdest* Pipelined Long-Integer Add
rdest ← last stage Graphics result
 last stage Graphics result ← *src1* + *src2*
- pfisub.w** *src1, src2, rdest* Pipelined Long-Integer Subtract
rdest ← last stage Graphics result
 last stage Graphics result ← *src1* - *src2*
- pfix.p** *src1, rdest* Pipelined Floating-Point to Integer Conversion
rdest ← last stage Adder result
 Advance A pipeline one stage
 A pipeline first stage ← 64-bit value with low-order 32 bits
 equal to integer part of *src1* rounded
- Pipelined Floating-Point Load**
- pfld.z** *src1(src2), freg* (Normal!)
- pfld.z** *src1(src2) + +, freg* (Autoincrement)
freg ← mem.z (third previous **pfld**'s (*src1* + *src2*))
 (where .z is precision of third previous **pfld.z**)
 If autoincrement
 THEN *src2* ← *src1* + *src2*
 FI
- pfle.p** *src1, src2, rdest* Pipelined F-P Less-Than or Equal Compare
 Assembler pseudo-operation, identical to **pfgt.p** except that
 assembler sets R-bit of instruction.
rdest ← last stage Adder result
 CC clear if *src1* ≤ *src2*, else set
 Advance A pipeline one stage
 A pipeline first stage is undefined, but no result exception occurs
- pfmam.p** *src1, src2, rdest* Pipelined Floating-Point Add and Multiply
rdest ← last stage Multiplier result
 Advance A and M pipeline one stage (operands accessed before advancing pipeline)
 A pipeline first stage ← A-op1 - A-op2
 M pipeline first stage ← M-op1 × M-op2

pfmov.r *src1, rdest* **Pipelined Floating-Point Reg-Reg Move**
 Assembler pseudo-operation
pfmov.ss *src1, rdest* = **pfadd.ss** *src1, f0, rdest*
pfmov.dd *src1, rdest* = **pfadd.dd** *src1, f0, rdest*
pfmov.sd *src1, rdest* = **pfamov.sd** *src1, rdest*
pfmov.ds *src1, rdest* = **pfamov.ds** *src1, rdest*

pfmsm.p *src1, src2, rdest* **Pipelined Floating-Point Subtract and Multiply**
rdest ← last stage Multiplier result
 Advance A and M pipeline one stage (operands accessed before advancing pipeline)
 A pipeline first stage ← A-op1 – A-op2
 M pipeline first stage ← M-op1 × M-op2

pfmul.p *src1, src2, rdest* **Pipelined Floating-Point Multiply**
rdest ← last stage Multiplier result
 Advance M pipeline one stage
 M pipeline first stage ← *src1* × *src2*

pfmul3.p *src1, src2, rdest* **Three-Stage Pipelined Multiply**
rdest ← last stage Multiplier result
 Advance 3-Stage M pipeline one stage
 M pipeline first stage ← *src1* × *src2*

pform *src1, rdest* **Pipelined OR to MERGE Register**
rdest ← last stage Graphics result
 last stage Graphics result ← *src1* OR MERGE
 MERGE ← 0

pfsm.p *src1, src2, rdest* **Pipelined Floating-Point Subtract and Multiply**
rdest ← last stage Adder result
 Advance A and M pipeline one stage (operands accessed before advancing pipeline)
 A pipeline first stage ← A-op1 – A-op2
 M pipeline first stage ← M-op1 × M-op2

pfsub.p *src1, src2, rdest* **Pipelined Floating-Point Subtract**
rdest ← last stage Adder result
 Advance A pipeline one stage
 A pipeline first stage ← *src1* + *src2*

pftrunc.p *src1, rdest* **Pipelined Floating-Point to Integer Conversion**
rdest ← last stage Adder result
 Advance A pipeline one stage
 A pipeline first stage ← 64-bit value with low-order 32 bits
 equal to integer part of *src1*

pfzchk1 *src1, src2, rdest* **Pipelined 32-Bit Z-Buffer Check**
 Consider *src1*, *src2*, and *rdest*, as arrays of two 32-bit
 fields *src1*(0)..*src1*(1), *src2*(0)..*src2*(1), and *rdest*(0)..*rdest*(1)
 where zero denotes the least significant field.
 PM ← PM shifted right by 2 bits
 FOR i = 0 to 1
 DO
 PM [i + 6] ← *src2*(i) ≤ *src1*(i) (unsigned)
 rdest(i) ← last stage Graphics result
 last stage Graphics result ← smaller of *src2*(i) and *src1*(i)
 OD
 MERGE ← 0

pfzchk	<i>src1, src2, rdest</i>	Pipelined 16-Bit Z-Buffer Check
	Consider <i>src1</i> , <i>src2</i> , and <i>rdest</i> , as arrays of four 16-bit fields <i>src1</i> (0).. <i>src1</i> (3), <i>src2</i> (0).. <i>src2</i> (3), and <i>rdest</i> (0).. <i>rdest</i> (3) where zero denotes the least significant field.	
	PM ← PM shifted right by 4 bits	
	FOR <i>i</i> = 0 to 3	
	DO	
	PM [<i>i</i> + 4] ← <i>src2</i> (<i>i</i>) ≤ <i>src1</i> (<i>i</i>) (unsigned)	
	<i>rdest</i> (<i>i</i>) ← last stage Graphics result	
	last stage Graphics result ← smaller of <i>src2</i> (<i>i</i>) and <i>src1</i> (<i>i</i>)	
	OD	
	MERGE ← 0	
pst.d	<i>freg, #const(src2)</i>	Pixel Store
pst.d	<i>freg, #const(src2) +</i>	Pixel Store Autoincrement
	Pixels enabled by PM in mem.D (<i>src2</i> + # <i>const</i>) ← <i>freg</i>	
	Shift PM right by 8/pixel size (in bytes) bits	
	IF autoincrement	
	THEN <i>src2</i> ← # <i>const</i> + <i>src2</i>	
	FI	
shl	<i>src1, src2, rdest</i>	Shift Left
	<i>rdest</i> ← <i>src2</i> shifted left by <i>src1</i> bits	
shr	<i>src1, src2, rdest</i>	Shift Right
	SC (in <i>psr</i>) ← <i>src1</i>	
	<i>rdest</i> ← <i>src2</i> shifted right by <i>src1</i> bits	
shra	<i>src1, src2, rdest</i>	Shift Right Arithmetic
	<i>rdest</i> ← <i>src2</i> arithmetically shifted right by <i>src1</i> bits	
shrd	<i>src1, src2, rdest</i>	Shift Right Double
	<i>rdest</i> ← low-order 32 bits of <i>src1:src2</i> shifted right by SC bits	
st.c	<i>src1ni, ctrlreg</i>	Store to Control Register
	<i>ctrlreg</i> ← <i>src1ni</i>	
st.x	<i>src1ni, #const(src2)</i>	Store Integer
	<i>mem.x (src2 + #const)</i> ← <i>src1ni</i>	
subs	<i>src1, src2, rdest</i>	Subtract Signed
	<i>rdest</i> ← <i>src1</i> - <i>src2</i>	
	OF ← (bit 31 carry ≠ bit 30 carry)	
	CC set if <i>src2</i> > <i>src1</i> (signed)	
	CC clear if <i>src2</i> ≤ <i>src1</i> (signed)	
subu	<i>src1, src2, rdest</i>	Subtract Unsigned
	<i>rdest</i> ← <i>src1</i> - <i>src2</i>	
	OF ← NOT (bit 31 carry)	
	CC ← bit 31 carry	
	(i.e. CC set if <i>src2</i> ≤ <i>src1</i> (unsigned)	
	CC clear if <i>src2</i> > <i>src1</i> (unsigned)	
trap	<i>src1, src2, rdest</i>	Software Trap
	Generate trap with IT set in <i>psr</i>	
unlock	End Interlocked Sequence
	Clear BL in <i>dirbase</i> . The next load or store unlocks the bus.	
	Enable interrupts after bus is unlocked.	
xor	<i>src1, src2, rdest</i>	Logical Exclusive OR
	<i>rdest</i> ← <i>src1</i> XOR <i>src2</i>	
	CC set if result is zero, cleared otherwise	
xorh	# <i>const, src2, rdest</i>	Logical Exclusive OR High
	<i>rdest</i> ← (# <i>const</i> shifted left 16 bit) XOR <i>src2</i>	
	CC set if result is zero, cleared otherwise	

Table 8.2. FADDP MERGE Update

Pixel Size (from PS)	Fields Loaded From Result into MERGE	Right Shift Amount (Field Size)
8	63..56, 47..40, 31..24, 15..8	8
16	63..58, 47..42, 31..26, 15..10	6
32	63..56, 31..24	8

8.2 Instruction Format and Encoding

All instructions are one word long and begin on a word boundary. When operands are registers, the register encodings shown in Table 8.3 are used. There are two general core-instruction formats, REG-format and CTRL-format, as well as a separate format for floating-point instructions.

8.2.1 REG-FORMAT INSTRUCTIONS

Within the REG-format are several variations as shown in Figure 8.1. Table 8.4 gives the encodings for these instructions. One encoding is an escape code that defines yet another variation: the core escape instructions. Figure 8.2 shows the format of this group, and Table 8.5 shows the encodings.

In these instructions, the *src2* field selects one of the 32 integer registers (most instructions) or five control registers (**st.c** and **ld.c**). *Dest* selects one of the 32 integer registers (most instructions) or floating-point registers (**fld**, **fst**, **pfld**, **pst**, **ixfr**). For instructions where *src1* is optionally an immediate value, bit 26 of the opcode (I-bit) indicates whether *src1* is an immediate. If bit 26 is clear, an integer register is used; if bit 26 is set, *src1* is contained in the low-order 16 bits, except for **bte** and **btne** instructions. For **bte** and **btne**, the five-bit immediate value is contained in the *src1* field. For **st**, **bte**, **btne**, and **bla**, the upper five bits of the *offset* or *broffset* are contained in the *dest* field instead of *src1*, and the lower 11 bits of *offset* are the lower 11 bits of the instruction.

Table 8.3. Register Encoding

Register	Encoding
r0	0
.	.
.	.
.	.
r31	31
f0	0
.	.
.	.
.	.
f31	31
Fault Instruction	0
Processor Status	1
Directory Base	2
Data Breakpoint	3
Floating-Point Status	4
Extended Process Status	5

For **ld** and **st**, bits 28 and zero determine operand size as follows:

Bit 28	Bit 0	Operand Size
0	0	8-bits
0	1	8-bits
1	0	16-bits
1	1	32-bits

When *src1* is an immediate and bit 28 is set, bit zero of the immediate value is forced to zero.

For **fld**, **fst**, **pfld**, **pst**, and **flush**, bit 0 selects autoincrement addressing if set. Bits one and two select the operand size as follows:

Bit 1	Bit 2	Operand Size
0	0	64-bits
0	1	128-bits
1	0	32-bits
1	1	32-bits

When *src1* is an immediate value, bits zero and one of the immediate value are forced to zero to maintain alignment. When bit one of the immediate value is clear, bit two is also forced to zero.

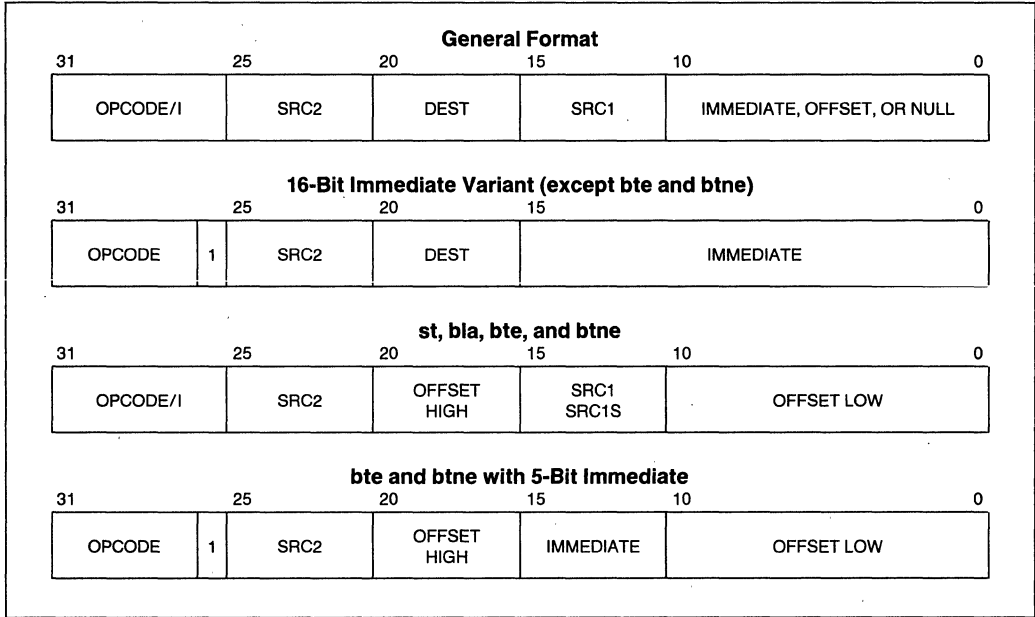


Figure 8.1. REG-Format Variations

Table 8.4. REG-Format Opcodes

		31			26		
ld.x	Load Integer	0	0	0	L	0	I
st.x	Store Integer	0	0	0	L	1	1
ixfr	Integer to F-P Reg Transfer	0	0	0	0	1	0
	(reserved)	0	0	0	1	1	0
fld.x, fst.x	Load/Store F-P	0	0	1	0	LS	I
flush	Flush	0	0	1	1	0	1
pst.d	Pixel Store	0	0	1	1	1	1
ld.c, st.c	Load/Store Control Register	0	0	1	1	LS	0
bri	Branch Indirect	0	1	0	0	0	0
trap	Trap	0	1	0	0	0	1
	(Escape for F-P Unit)	0	1	0	0	1	0
	(Escape for Core Unit)	0	1	0	0	1	1
bte, btne	Branch Equal or Not Equal	0	1	0	1	E	I
pfld.y	Pipelined F-P Load	0	1	1	0	0	I
	(CTRL-Format Instructions)	0	1	1	x	x	x
addu, -s, subu, -s,	Add/Subtract	1	0	0	SO	AS	I
shl, shr	Logical Shift	1	0	1	0	LR	I
shrd	Double Shift	1	0	1	1	0	0
bla	Branch LCC Set and Add	1	0	1	1	0	1
shra	Arithmetic Shift	1	0	1	1	1	I
and(h)	AND	1	1	0	0	H	I
andnot(h)	ANDNOT	1	1	0	1	H	I
or(h)	OR	1	1	1	0	H	I
xor(h)	XOR	1	1	1	1	H	I
	(reserved)	1	1	x	x	1	0

- L Integer Length
 - 0 —8 bits
 - 1 —16 or 32 bits (selected by bit 0)
- LS Load/Store
 - 0 —Load
 - 1 —Store
- SO Signed/Ordinal
 - 0 —Ordinal
 - 1 —Signed
- H High
 - 0 —and, or, andnot, xor
 - 1 —andh, orh, andnoth, xorh

- AS Add/Subtract
 - 0 —Add
 - 1 —Subtract
- LR Left/Right
 - 0 —Left Shift
 - 1 —Right Shift
- E Equal
 - 0 —Branch on Not Equal
 - 1 —Branch on Equal
- I Immediate
 - 0 —src1 is register
 - 1 —src1 is immediate

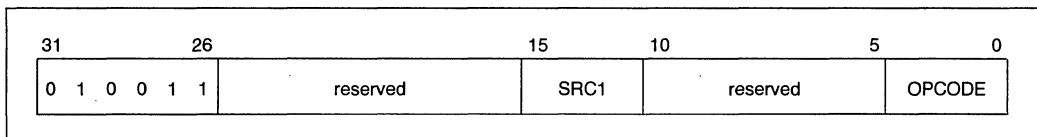


Figure 8.2. Core Escape Instruction Format

Table 8.5. Core Escape Opcodes

		4	0			
	(reserved)	0	0	0	0	0
lock	Begin Interlocked Sequence	0	0	0	0	1
calli	Indirect Subroutine Call	0	0	0	1	0
	(reserved)	0	0	0	1	1
intovr	Trap on Integer Overflow	0	0	1	0	0
	(reserved)	0	0	1	0	1
	(reserved)	0	0	1	1	0
unlock	End Interlocked Sequence	0	0	1	1	1
	(reserved)	0	1	x	x	x
	(reserved)	1	0	x	x	x
	(reserved)	1	1	x	x	x

8.2.2 CTRL-FORMAT INSTRUCTIONS

The CTRL instructions do not refer to registers, so instead of the register fields, they have a 26-bit relative branch offset. Figure 8.3 shows the format of these instructions and Table 8.6 defines the encodings.

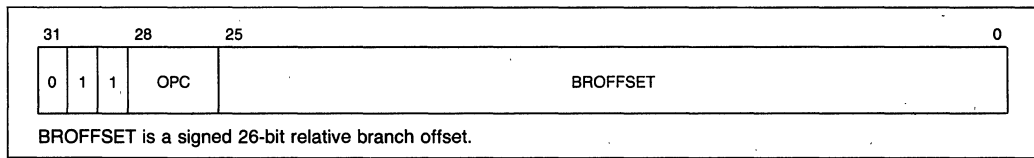


Figure 8.3. CTRL Instruction Format

Table 8.6. CTRL-Format Opcodes

		28	26	
br	Branch Direct	0	1	0
call	Call	0	1	1
bc(t)	Branch on CC Set	1	0	T
bnc(t)	Branch on CC Clear	1	1	T

T Taken
 0 —bc or bnc
 1 —bc.t or bnc.t

8.2.3 FLOATING-POINT INSTRUCTIONS

The floating-point instructions also constitute an escape series. All these instructions begin with the bit sequence 010010. Figure 8.4 shows the format of the floating point instructions, and Table 8.7 gives the encodings. Within the dual-operation instructions is a subcode DPC whose values are given in Table 8.8 along with the mnemonic that corresponds to each.

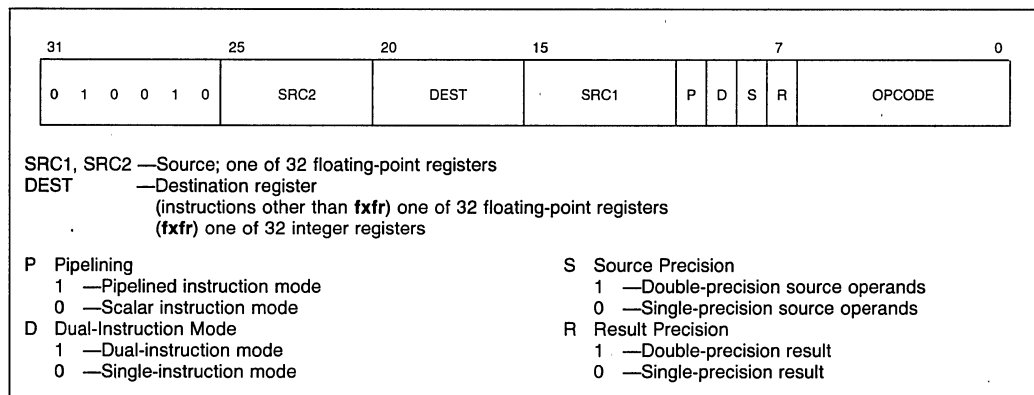


Figure 8.4. Floating-Point Instruction Encoding

Table 8.7. Floating-Point Opcodes

		6					0
pfam	Add and Multiply*						
pfmam	Multiply with Add*	0	0	0			DPC
pfsm	Subtract and Multiply*	0	0	1			DPC
pfmsm	Multiply with Subtract*						
(p)fmul	Multiply	0	1	0	0	0	0
fmlow	Multiply Low	0	1	0	0	0	1
frcp	Reciprocal	0	1	0	0	0	1
frsqr	Reciprocal Square Root	0	1	0	0	0	1
(p)fadd	Add	0	1	1	0	0	0
(p)fsub	Subtract	0	1	1	0	0	1
(p)fix	Fix	0	1	1	0	0	1
(p)famov	Adder Move	0	1	1	0	0	1
pfgt/pfle**	Greater Than	0	1	1	0	1	0
pfeq	Equal	0	1	1	0	1	1
(p)ftrunc	Truncate	0	1	1	1	0	1
fxfr	Transfer to Integer Register	1	0	0	0	0	0
(p)fiadd	Long-Integer Add	1	0	0	1	0	1
(p)fisub	Long-Integer Subtract	1	0	0	1	1	1
(p)zfchkl	Z-Check Long	1	0	1	0	1	1
(p)zfchks	Z-Check Short	1	0	1	1	1	1
(p)faddp	Add with Pixel Merge	1	0	1	0	0	0
(p)faddz	Add with Z Merge	1	0	1	0	0	1
(p)form	OR with MERGE Register	1	0	1	1	0	1

*pfam and pfsm have P-bit set; pfmam and pfmsm have P-bit clear.

**pfgt has R bit cleared; pfle has R bit set.

The following table shows the opcode mnemonics that generate the various encodings of DPC and explains each encoding.

Table 8.8. DPC Encoding

DPC	PFAM Mnemonic	PFMS Mnemonic	M-Unit op1	M-Unit op2	A-Unit op1	A-Unit op2	T Load	K Load*
0000	r2p1	r2s1	KR	src2	src1	M result	No	No
0001	r2pt	r2st	KR	src2	T	M result	No	Yes
0010	r2ap1	r2as1	KR	src2	src1	A result	Yes	No
0011	r2apt	r2ast	KR	src2	T	A result	Yes	Yes
0100	i2p1	i2s1	KI	src2	src1	M result	No	No
0101	i2pt	i2st	KI	src2	T	M result	No	Yes
0110	i2ap1	i2as1	KI	src2	src1	A result	Yes	No
0111	i2apt	i2ast	KI	src2	T	A result	Yes	Yes
1000	rat1p2	rat1s2	KR	A result	src1	src2	Yes	No
1001	m12apm	m12asm	src1	src2	A result	M result	No	No
1010	ra1p2	ra1s2	KR	A result	src1	src2	No	No
1011	m12t1pa	m12t1sa	src1	src2	T	A result	Yes	No
1100	iat1p2	iat1s2	KI	A result	src1	src2	Yes	No
1101	m12tpm	m12t1sm	src1	src2	T	M result	No	No
1110	ia1p2	ia1s2	KI	A result	src1	src2	No	No
1111	m12t1pa	m12t1sa	src1	src2	T	A result	No	No

DPC	PFAM Mnemonic	PFMS Mnemonic	M-Unit op1	M-Unit op2	A-Unit op1	A-Unit op2	T Load	K Load*
0000	mr2p1	mr2s1	KR	src2	src1	M result	No	No
0001	mr2pt	mr2st	KR	src2	T	M result	No	Yes
0010	mr2mp1	mr2ms1	KR	src2	src1	M result	Yes	No
0011	mr2mpt	mr2mst	KR	src2	T	M result	Yes	Yes
0100	mi2p1	mi2s1	KI	src2	src1	M result	No	No
0101	mi2pt	mi2st	KI	src2	T	M result	No	Yes
0110	mi2mp1	mi2ms1	KI	src2	src1	M result	Yes	No
0111	mi2mpt	mi2mst	KI	src2	T	M result	Yes	Yes
1000	mrmt1p2	mrmt1s2	KR	M result	src1	src2	Yes	No
1001	mm12mpm	mm12msm	src1	src2	M result	M result	No	No
1010	mrm1p2	mrm1s2	KR	M result	src1	src2	No	No
1011	mm12t1pm	mm12t1t1sm	src1	src2	T	A result	Yes	No
1100	mimt1p2	mimt1s2	KI	M result	src1	src2	Yes	No
1101	mm12tpm	mm12t1sm	src1	src2	T	M result	No	No
1110	mim1p2	mim1s2	KI	M result	src1	src2	No	No
1111								

Intel-Reserved

*If K-load is set, KR is loaded when operand-1 of the multiplier is KR; KI is loaded when operand-1 of the multiplier is KI.

8.3 Instruction Timings

i860 microprocessor instructions take one clock to execute unless a freeze condition is invoked. Freeze conditions and their associated delays are shown in

the table below. Freezes due to multiple simultaneous cache misses result in a delay that is the sum of the delays for processing each miss by itself. Other multiple freeze conditions usually add only the delay of the longest individual freeze.

Freeze Condition	Delay
Instruction-cache miss	Number of clocks to read instruction (from ADS clock to first READY # clock) plus time to last READY # of block when jump or freeze occurs during miss processing plus two clocks if data-cache being accessed when instruction-cache miss occurs.
Reference to destination of ld instruction that misses	One plus number of clocks to read data (from ADS# clock to first READY # clock) minus number of instructions executed since load (not counting instruction that references load destination)
fld miss	One plus number of clocks from ADS# to first READY #
call/calli/ixfr/fixr/ld.c/st.c and data cache miss processing in progress	One plus number of clocks until first READY # returned
ld/st/pfld/fld/fst and data cache miss processing in progress	One plus number of clocks until last READY # returned
Reference to <i>dest</i> of ld , call , calli , fixr , or ld.c in the next instruction	One clock

Freeze Condition	Delay
Reference to <i>dest</i> of fld/plfd/ixfr in the next two instructions	Two clocks in the first instruction; one in the second instruction
bc/bnc/bc.t/bnc.t following fadd/fsub/pfeg/pfgt	One clock
<i>Src1</i> of multiplier operation refers to result of previous operation	One clock
Floating-point or graphics unit instruction or fst , and scalar operation in progress other than frcp or frsqr	If the scalar operation is fadd, fix, fmlow, fmul.ss, fmul.sd, ftrunc, or fsub , two minus the number of instructions already executed after the scalar operation. If the scalar operation is fmul.dd , three minus the number of instructions executed after it. Add one if the precision of the result of the previous scalar operation is different than that of the source. Add one if the floating-point operation is pipelined and its destination is not f0 . If the sum of the above terms is negative, there is no delay.
Multiplier operation preceded by a double-precision multiply	One clock
Multiplier operation with data pattern requiring extra rounding operation	One clock
TLB miss	Five plus the number of clocks to finish two reads plus the number of clocks to set A-bits (if necessary)
pfld when three pfld 's are outstanding	One plus the number of clocks to return data from first pfld
pfld hits in the data cache	Two plus the number of clocks to finish all outstanding accesses
Store pipe full (two internal plus outstanding bus cycles) and st/fst miss, ld miss, or flush with modified block	One plus the number of clocks until READY # active on next write data
Address pipe full (one internal plus outstanding bus cycles) and ld/fld/plfd/st/fst	Number of clocks until next address can be issued
ld/fld following st/fst hit	One clock
Delayed branch not taken	One clock
Nondelayed branch taken: bc, bnc bte, btne	One clock Two clocks
Branch indirect bri	One clock
st.c	Two clocks
Result of graphics-unit instruction (other than fmov) used in next instruction when the next instruction is an adder- or multiplier-unit instruction	One clock
Result of graphics-unit instruction used in next instruction when the next instruction is a graphics-unit instruction	One clock
flush followed by flush	Three clocks
fst followed by pipelined floating-point operation that overwrites the register being stored	One clock

8.4 Instruction Characteristics

The following table lists some of the characteristics of each instruction. The characteristics are:

- What processing unit executes the instruction. The codes for processing units are:
 - A Floating-point adder unit
 - E Core execution unit
 - G Graphics unit
 - M Floating-point multiplier unit
- Whether the instruction is pipelined or not. A *P* indicates that the instruction is pipelined.
- Whether the instruction is a delayed branch instruction. A *D* marks the delayed branches.
- Whether the instruction changes the condition code CC. A *CC* marks those instructions that change CC.
- Which faults can be caused by the instruction. The codes used for exceptions are:

IT	Instruction Fault
SE	Floating-Point Source Exception
RE	Floating-Point Result Exception, including overflow, underflow, inexact result
DAT	Data Access Fault

The instruction access fault IAT and the interrupt trap IN are not shown in the table because they can occur for any instruction.

- Performance notes. These comments regarding optimum performance are recommendations only. If these recommendations are not followed, the i860 microprocessor automatically waits the necessary number of clocks to satisfy internal hardware requirements. The following notes define the numeric codes that appear in the instruction table:
 1. The following instruction should not be a conditional branch (**bc**, **bnc**, **bc.t**, or **bnc.t**).
 2. The destination should not be a source operand of the next two instructions.
 3. A load should not directly follow a store that is expected to hit in the data cache.
 4. When the prior instruction is scalar, *src1* should not be the same as the *rdest* of the prior operation.

5. The *freg* should not reference the destination of the next instruction if that instruction is a pipelined floating-point operation.
 6. The destination should not be a source operand of the next instruction.
 7. When the prior operation is scalar and multiplier *op1* is *src1*, *src2* should not be the same as the *rdest* of the prior operation.
 8. When the prior operation is scalar, *src1* and *src2* of the current operation should not be the same as *rdest* of the prior operation.
 9. A **pfld** should not immediately follow a **pfld**.
- Programming restrictions. These indicate combinations of conditions that must be avoided by programmers, assemblers, and compilers. The following notes define the alphabetic codes that appear in the instruction table:
 - a. The sequential instruction following a delayed control-transfer instruction may not be another control-transfer instruction (except in the case of external interrupts), nor a trap instruction, nor the target of a control-transfer instruction.
 - b. When using a **bri** to return from a trap handler, programmers should take care to prevent traps from occurring on that or on the next sequential instruction. IM should be zero (interrupts disabled) when the **bri** is executed.
 - c. If *rdest* is not zero, *src1* must not be the same as *rdest*.
 - d. When *src1* goes to the multiplier *op1*, KR, or KI, *src1* must not be the same as *rdest*.
 - e. If *rdest* is not zero, *src1* and *src2* must not be the same as *rdest*.
 - f. *src1* must not be the same as *src2* for the autoincrementing form of this instruction.

Table 8.9 Instruction Characteristics

Instruction	Execution Unit	Pipelined? Delayed?	Sets CC?	Faults	Performance Notes	Programming Restrictions
adds	E		CC		1	
addu	E		CC		1	
and	E		CC			
andh	E		CC			
andnot	E		CC			
andnoth	E		CC			
bc	E					
bc.t	E	D				a
bla	E	D				a, f
bnc	E					
bnc.t	E	D				a
br	E	D				a
bri	E	D				a, b
bte	E					
btne	E					
call	E	D			6	a
calli	E	D			6	a
fadd.p	A			SE, RE		
faddp	G				8	
faddz	G				8	
famov.r	A			SE		
fiadd.z	G				8	
fisub.z	G				8	
fix.p	A			SE, RE		
fid.y	E			DAT	2, 3	f
flush	E					
fmlow.p	M				4	
fmul.p	M			SE, RE	4	
form	G				8	
frep.p	M			SE, RE		
frsqr.p	M			SE, RE		
fst.y	E			DAT	5	f
fsub.p	A			SE, RE		
ftrunc.p	A			SE, RE		
fxfr	G				6, 8	
fzchkl	G				8	
fzchks	G				8	
intovr	E			IT		
ixfr	E				2	
ld.c	E					
ld.x	E			DAT	6	
or	E		CC			
orh	E		CC			
pfadd.p	A	P		SE, RE		
pfaddp	G	P			8	e

Instruction	Execution Unit	Pipelined? Delayed?	Sets CC?	Faults	Performance Notes	Programming Restrictions
pfaddz	G	P			8	e
pfam.p	A&M	P		SE, RE	7	d
pfamov.r	A	P		SE		
pfreq.p	A	P	CC	SE	1	
pfgt.p	A	P	CC	SE	1	
pfiaadd.z	G	P			8	e
pfisub.z	G	P			8	e
pfix.p	A	P		SE, RE		
pfld.z	E	P		DAT	2, 9	f
pfmam.p	A & M	P		SE, RE	7	d
pfmsm.p	A & M	P		SE, RE	7	d
pfmul.p	M	P		SE, RE	4	c
pform	G	P			8	e
pfsm.p	A&M	P		SE, RE	7	d
pfsub.p	A	P		SE, RE		
pftrunc.p	A	P		SE, RE		
pfzchkl	G	P			8	
pfzchks	G	P			8	
pst.d	E			DAT		f
shl	E					
shr	E					
shra	E					
shrd	E					
st.c	E					
st.x	E			DAT		
subs	E		CC		1	
subu	E		CC		1	
trap	E			IT		
xor	E		CC			
xorh	E		CC			

DATA SHEET REVISION REVIEW

The following list represents the key differences between version 002 and version 001 of the i860 Microprocessor Data Sheet.

1. Big-endian description in section 2.3 has been expanded.
2. Bit 17 of the Extended Processor Status Register (EPSR) is the INT bit which reflects the value on the interrupt pin (INT), as described in section 2.2.4 entitled "EXTENDED PROCESSOR STATUS REGISTER". This is a documentation update only.
3. The cacheability of a page is controlled by NOR'ing the value of the CD, WT bits and the KEN# input pin, as described in section 2.5 entitled "Caching and Cache Flushing" and section 3.1.14 entitled "Cache Enable (KEN#)". This is a documentation update only.
4. The NOTE section in section 2.5 entitled "Caching and Cache Flushing" has been updated to clarify the paging requirement on changing the DTB field in the **dirbase** register.
5. Information on register encoding is added in section 8.2 entitled "Instruction Format and Encoding". This is a documentation update only.

The following list represents the key differences between this version and version 002 of the i860 Microprocessor Data Sheet.

Specification Changes:

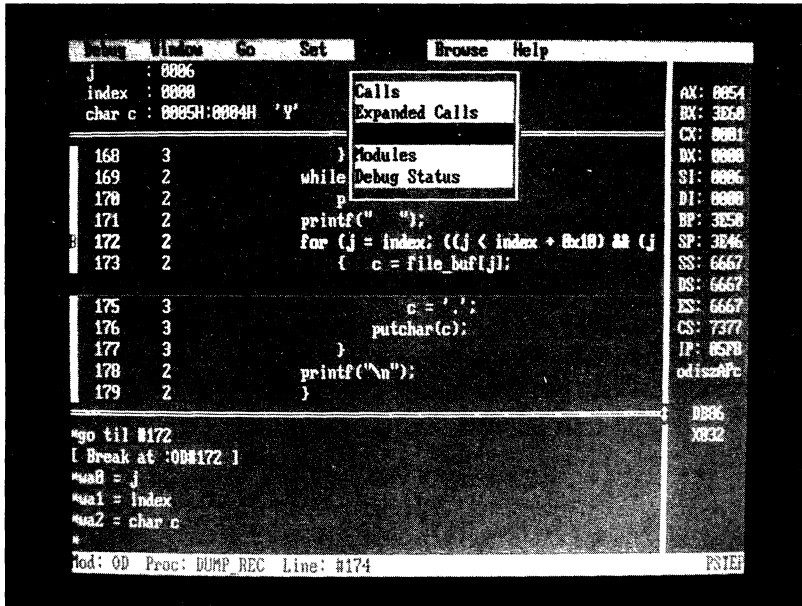
1. Specification changes for improved AC performance are in section 7.3.
2. HOLD is acknowledged during locked bus cycles. See section 3.1.8.
3. Additional paths have been added to the bus state diagram to allow direct transitions from states T12 and T11 to state TH. See Figures 4.1 and 4.10.
4. Two new instructions, **(p)famov.r**, have been added. These replace **(p)fadd.ds** and **(p)fadd.sd** in the assembler pseudo-ops **(p)fmov.r**. These changes are in section 8.1 and tables 2.7, 8.7, and 8.9.

Documentation Changes:

1. Big and little endian description has been expanded in sections 2.2.2, 2.3, and Figure 2.8.
2. The actions and explanations of the **lock**, **unlock**, and **st.c dirbase** changing the BL bit have been updated in sections 2.2.4, 3.1.5, 3.1.8, 4.3.4, 4.3.5, and 8.1.
3. The explanation of the AA and MA bits of the **fpwr** have been expanded in section 2.2.8.
4. The explanation of the WT bit of the Page Table Entries has been expanded in sections 2.4.4.4 and 2.5.
5. A change concerning the locking of the bus during address translation is explained in sections 2.4.5 and 2.8.5.
6. A further explanation on when to flush the data cache is given in section 2.5.
7. The explanation of the floating point multiplier pipeline has been expanded in section 2.6.1.
8. The explanation of BREQ has been expanded in section 3.1.4 and Figure 4.1.
9. The explanation of result exceptions has been expanded in sections 2.8 and 3.2.
10. Instruction fetch identification has been clarified in section 3.1.6 and table 3.2.
11. Bus cycle diagrams in Figures 4.7, 4.8, and 4.10 have been clarified/corrected.
12. Precision specification **.r** has been added to section 8.0 and table 8.1.
13. In section 8.4, performance note 9 has been added, programming restriction **d** has been changed, and programming restriction **f** has been added. Table 8.9 has been updated to reflect these changes.
14. The description of testability has changed in sections 3.3. and 3.3.2. RESET and HOLD must be asserted by the tester to force the chip outputs to float (tri-state).

**Development Tools for the
8086, 80186, 80188, 80286,
80386, and 80486**

6



COMPLETE SOFTWARE DEVELOPMENT SUPPORT FOR THE 8086/80186 FAMILY OF MICROPROCESSORS

Intel supports application development for the 8086/80186 family of microprocessors (8086, 8088, 80186, 80188 and real mode 80286 and 80386 designs) with a complete set of development languages and utilities. These tools include a macro assembler and compilers for C, PL/M, FORTRAN and Pascal. A linker/relocator program, library manager, numerics support libraries, and object-to-hex utility are also available. Intel software tools generate fast and efficient code. They are designed to give maximum control over the processor. Most importantly, they are designed to get your application up and running in an embedded system fast and with maximum design productivity.

FEATURES

- Macro assembler for speed-critical code
- NEW windowed, interactive source level debugger works with all Intel languages
- ANSI Compatible iC-86 package for structured C programming, with many processor specific extensions
- PL/M compiler for high-level language programs with support for many low-level hardware functions
- FORTRAN for ANSI-compatible, numeric intensive applications
- Pascal for developing modular, portable applications that are easy to maintain
- Linker program for linking modules generated by Intel compilers and assemblers
- Locator for generating programs with absolute addresses for execution from ROM based systems
- AEDIT Source Code and text editor
- Library manager for creating and maintaining object module libraries
- Complete numeric support libraries including a software emulator for the 8087
- Object-to-hex conversion utility for burning code into (E)PROMS
- Hosted on IBM PC XT/AT* or compatibles running DOS, DEC VAX* or MicroVAX* systems running VMS, and Intel Systems 86/3XX or 286/3XX running iRMX® Operating System



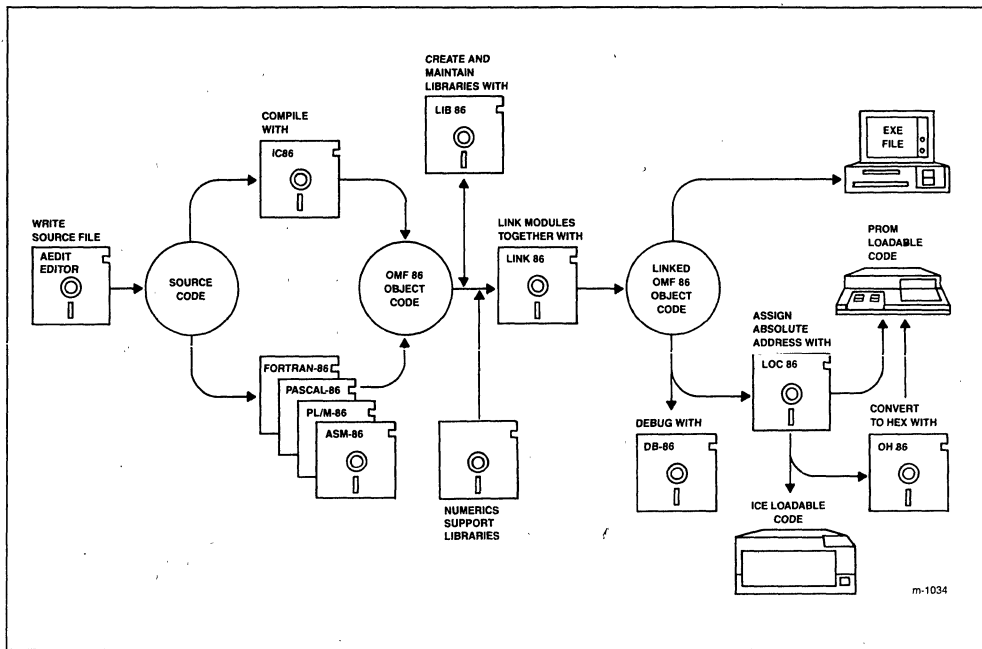


Figure 1: The Application Development Process

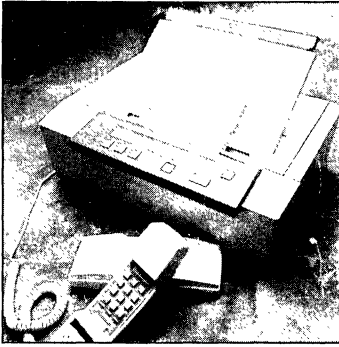
ASM-86 MACRO ASSEMBLER

ASM-86 is the macro assembler for the 8086/80186 family of components. It is used to translate symbolic assembly language source into relocatable object code where utmost speed, small code size and hardware control are critical. Intel's exclusive macro facility in ASM-86 saves development and maintenance time, since common code sequences need only be developed once. The assembler's simplified instruction set reduces the number of mnemonics that the programmer needs to remember. This assembler also saves development time by performing extensive checks on consistent usage of variables and labels. Inconsistencies are detected when the program is assembled, before linking or debugging is started.

NEW FOR 1989: SOURCE LEVEL DEBUGGER

DB-86 is an on-host software execution environment with source level debug capabilities for object modules produced by IC-86, ASM-86, PL/M-86, Pascal-86 and FORTRAN-86. Its powerful, source-oriented interface allows users to focus their efforts on finding bugs rather than spending time learning and manipulating the debug environment.

- **Ease of learning.** Drop-down menus make the tool easy to learn for new or casual users. A command line interface is also provided for more complex problems.
- **Extensive debug modes.** Watch windows, conditional breakpoints (breakpoints triggered by program conditions), trace points, and fixed and temporary breakpoints can be set and modified as needed.
- **See into your program.** You can browse source and call stack, observe processor registers, output screen, and watch window variables accessed by either the pull down menu or by a single keystroke using function keys.
- **Full debug symbolics for maximum productivity.** The user need not know whether a variable is an unsigned integer, a real, or a structure; the debugger utilizes the wealth of variable typing information available in Intel languages to display program variables in their respective type formats.
- **Support for overlaid programs and the numeric coprocessor.**



iC-86 SOFTWARE PACKAGE

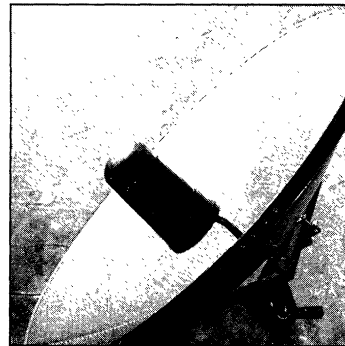
Intels iC-86 brings the full power of the C programming language to 8086, 8088, 80186 and 80188-based microprocessor systems. It can also be used to develop real mode programs for execution on the 80286 or 80386. iC-86 has been developed specifically for embedded microprocessor-based applications. iC-86 meets the draft proposed ANSI C standard. Key features of the iC-86 compiler include:

- **Highly Optimized.** Four levels of optimization are available. Important optimization features include a jump optimizer and improved register manipulation via register history.
- **ROMable Code and Libraries.** The iC-86 compiler produces ROMable code which can be loaded directly into embedded target systems. Libraries are also completely ROMable, retargetable and reentrant.
- **Supports Small, Medium, Compact, and Large memory segmentation models.**
- **Symbolics.** The iC-86 compiler boosts programming productivity by providing extensive debug information, including type information and symbols. The symbolics information can be used to debug using Intel ICE™ emulators and the new DB-86 Source level debugger.
- **Built-in functions.** iC-86 is loaded with built-in functions. The flags register, I/O ports, interrupts, and numerics chip can be controlled directly, without the need for assembly language coding. You spend more of your productive time programming in C and less with Assembler. Built-in functions also improve compile-time and run-time performance since the compiler generates in-line code instructions instead of function calls to assembly instructions.
- **Standard Language.** iC-86 conforms to the 1988 Draft Proposed ANSI standard for the C language. iC-86 code is fully linkable with other modules written in other Intel 8086/186 languages, allowing programmers to use the optimal language for any task.

PL/M-86 SOFTWARE PACKAGE

PL/M-86 is a high-level programming language designed to support the software requirements of advanced 16-bit microprocessors. PL/M-86 provides the productivity advantages of a high-level language while providing the low-level hardware access features of assembly language. Key features of PL/M-86 include:

- **Structured programming.** PL/M-86 supports modular and structured programming, making programs easier to understand, maintain and debug.
- **Built-in functions.** PL/M-86 includes an extensive list of functions, including TYPE CONVERSION functions, STRING manipulations, and functions for interrogating 8086/186 hardware flags.
- **Interrupt handling.** The INTERRUPT attribute allows you to define interrupt handling procedures. The compiler generates code to save and restore all registers for INTERRUPT procedures.
- **Compiler controls.** Compile-time options increase the flexibility of the PL/M-86 compiler. They include: optimization, conditional compilation, the inclusion of common PL/M source files from disk, cross-reference of symbols, and optional assembly language code in the listing file.
- **Data types.** PL/M-86 supports seven data types, allowing the compiler to perform three different kinds of arithmetic: signed, unsigned and floating point.
- **Language compatibility.** PL/M-86 object modules are compatible with all other object modules generated by Intel 8086/186 languages.



FORTRAN-86 SOFTWARE PACKAGE

FORTRAN-86 meets the ANSI FORTRAN 77 Language Subset Specification and includes almost all of the features of the full standard. This compatibility assures portability of existing FORTRAN programs and shortens the development process, since programmers are immediately productive without retraining.

FORTRAN-86 provides extensive support for numeric processing tasks and applications, with features such as:

- Support for single, double, double extended precision, complex, and double complex floating-point data types
- Support for proposed REALMATH IEEE floating point standard
- Full support for all other data types: integer, logical and character
- Optional hardware (8087 numeric data processor) or software (simulator) floating-point support at link time

PASCAL-86 SOFTWARE PACKAGE

Pascal-86 conforms to the ISO Pascal standard, facilitating application portability, training and maintenance. It has also been enhanced with microcomputer support features such as interrupt handling, direct port I/O and separate compilation.

A well-defined and documented run-time operating system interface allows the user to execute applications under user-designed operating systems as an alternate to the development system environment. Program modules compiled under Pascal-86 are compatible and linkable with modules written in other Intel 8086/186 languages, so developers can implement each module in the language most appropriate for the task at hand.

Pascal-86 object modules contain symbol and type information for program debugging using Intel ICE™ emulators and the DB-86 debugger.

LINK-86 LINKER

Intel's LINK-86 utility is used to combine multiple object modules into a single program and resolve references between independently compiled modules. The resulting linked module can be either a bound load-time-locatable module or simply a relocatable module. A .EXE option allows modules to be generated which can be executed directly on a DOS system.

LINK-86 greatly increases productivity by allowing you to use modular programming. The incremental link capability allows new modules to be easily added to existing software. Because applications can be broken into separate modules, they're easier to design, test and maintain. Standard modules can be reused in different applications, saving software development time.

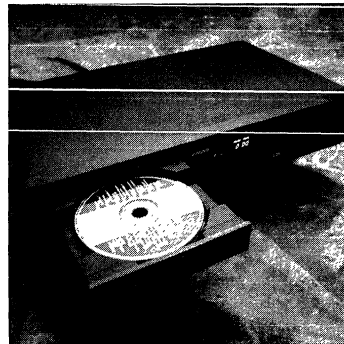
LOC-86 LOCATOR

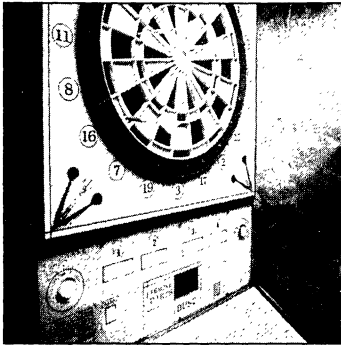
The LOC-86 utility changes relocatable 8086/186 object modules into absolute object modules. Its default address assignment algorithm will automatically assign absolute addresses to the object modules prior to loading of the code into the target system. This frees you from concern about the final arrangement of the object code in memory. You still have the power to override the control and specify absolute addresses for various Segments, Classes, and Groups in memory. You may also reserve various parts of memory.

LOC-86 is a powerful tool for embedded development because it simplifies set up of the bootstrap loader and initialization code for execution from ROM based systems. The locator will also optionally generate a print file containing diagnostic information to assist in program debugging.

NUMERICS SUPPORT LIBRARY

The Numerics Support Library greatly facilitates the use of floating-point calculations from programs written in Assembler, PL/M, and C. It adds to these languages many of the functions that are built into applications programming languages, such as Pascal and FORTRAN. A full 8087 software emulator and interface libraries are included for precision floating point calculations without the use of the 8087 component. The decimal conversion library aids the translation between decimal and binary formats. A Common Elementary Function library provides support for transcendental, rounding and other common functions, not directly handled by the numeric processor. An Error Handler Module makes it easy to write interrupt routines that recover from floating-point error conditions.





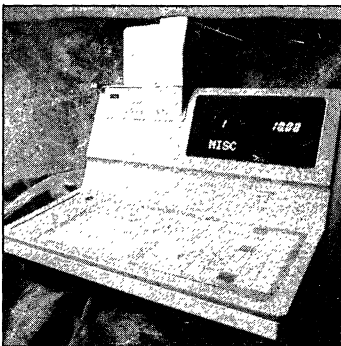
LIB-86 LIBRARIAN

The Intel LIB-86 utility creates and maintains libraries of software object modules. Standard modules can be placed in a library and linked to your application using the LINK-86 utility.

AEDIT SOURCE CODE AND TEXT EDITOR

AEDIT is a full-screen text editing system designed specifically for software engineers and technical writers. With the facilities for automatic program block indentation, HEX display and input, and full macro support, AEDIT is an essential tool for any programming environment. And with AEDIT, the output file is the pure ASCII text (or HEX code) you input—no special characters or proprietary formats.

Dual file editing means you can create source code and its supporting documents at the same time. Keep your program listing with its errors in the background for easy reference while correcting the source in the foreground. Using the split-screen windowing capability, it is easy to compare two files, or copy text from one to the other. The DOS system-escape command eliminates the need to leave the editor to compile a program, get a directory listing, or execute any other program executable at the DOS system level.

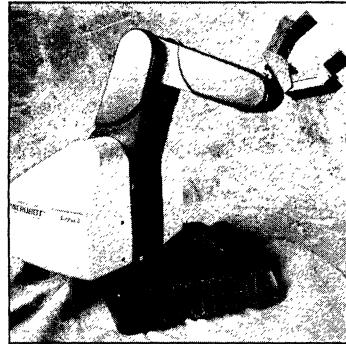


OH-86 OBJECT-TO-HEXADECIMAL CONVERTER

The OH-86 utility converts Intel 8086/186 object modules into standard hexadecimal format, allowing the code to be loaded directly into PROM using industry standard PROM programmers.

SERVICE, SUPPORT AND TRAINING

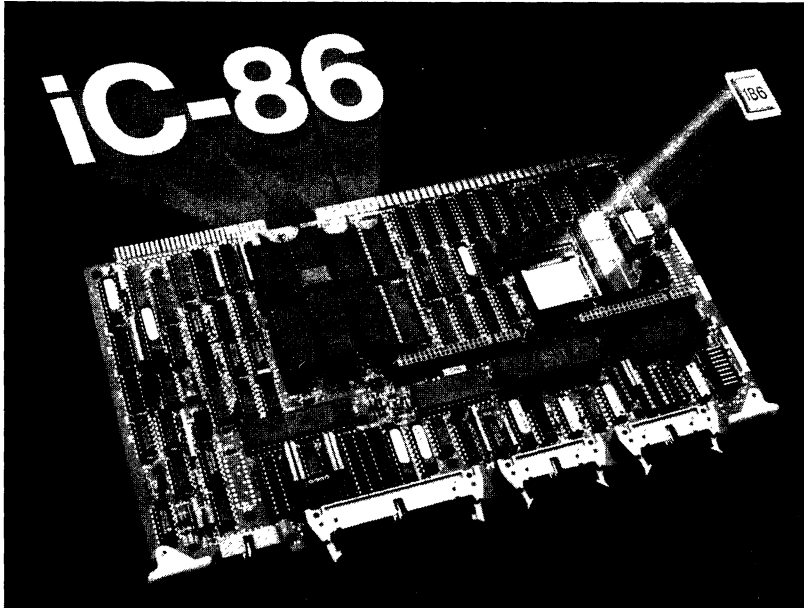
Intel augments its 8086/186 family development tools with a full array of seminars, classes and workshops. In addition, on-site consulting services, field application engineering expertise, telephone hotline support, and software and hardware maintenance contracts are available to help assure your design success.



ORDERING INFORMATION

D86ASM86NL	ASM-86	Assembler for PC XT or AT system (or compatible) running DOS 3.0 or higher	MVVSPLM86	PL/M-86	Software Package for MicroVAX/VMS
VVSASM86	ASM-86	Assembler for VAX/VMS	R86PLM86SU	PL/M-86	Software Package for Intel System 8086/3XX running iRMX 86 operating system
MVVSASM86	ASM-86	Assembler for MicroVAX/VMS	D86FOR86NL	FORTRAN-86	Software Package for PC XT/AT (or compatible) running PC-DOS 3.0 or higher
R86ASM86SU	ASM-86	Assembler for Intel 86/3XX systems running iRMX 86 operating system	VVSFORT86	FORTRAN-86	Software Package for VAX/VMS 4.3 and later
R286ASM86EU	ASM-86	Assembler for Intel 286/3XX systems running iRMX II™ operating system	MVVSFORT86	FORTRAN-86	Software Package for MicroVAX/VMS
Note:	ASM-86 includes Macro Assembler, Link-86, Loc-86, Lib-86, Cross-Reference utility, OH-86, Numerics Support, and DB-86 Source Level Debugger. (DB-86 available in DOS version only.)		R86FOR86SU	FORTRAN-86	Software Package for Intel System 86/3XX running iRMX 86 operating system
D86C86NL	iC-86	Software Package for IBM PC XT/AT running PC DOS 3.0 or higher	D86PAS86NL	PASCAL-86	Software Package for IBM PC XT/AT running PC DOS 3.0 or higher
VVSC86	iC-86	Software Package for VAX/VMS	VVSPAS86	PASCAL-86	Software Package for VAX/VMS
MVVS86	iC-86	Software Package for MicroVAX/VMS	MVVPAS86	PASCAL-86	Software Package for MicroVAX/VMS
R86C86SU	iC-86	Software Package for Intel System 8086/3XX running iRMX 86 operating system	R86PAS86SU	PASCAL-86	Software Package for Intel System 86/3XX running iRMX 86
D86PLM86NL	PL/M-86	Software Package for IBM PC XT/AT running PC DOS 3.0 or higher	D86EDNL		A&E:IT Source Code Editor for IBM PC XT/AT running PC DOS 3.0 or higher
VVSPLM86	PL/M-86	Software Package for VAX/VMS			

ICE is a trademark and iRMX a registered trademark of Intel Corporation.
 VAX and VMS are registered trademarks of Digital Equipment Corporation.
 IBM PC XT/AT is a trademark of International Business Machines Corporation.



INTEL iC-86/286 COMPILER

The Intel iC-86/286 compiler is the C compiler to use for 8086/186/286 embedded microprocessor designs. In addition to outstanding execution speed, Intel's iC-86/286 compiler generates compact, efficient code which can be easily loaded into ROM-based systems. The iC-86/286 compiler is also fully supported by the Intel DB86 windowed source-level software debugger and in-circuit emulation tools.

iC-86/286 COMPILER FEATURES

- Optimized for embedded systems
- Built-in functions for automatic machine code generation
- ROMable code and libraries
- Integrated debugging with Intel ICE™ and I²ICE™
- Compliance with draft ANSI standard
- Supports Small, Medium, Compact, and Large memory models
- PL/M compatible subsystems
- Selector data type support
- Linkable with other Intel 8086/286 languages such as ASM and PL/M
- ROMable and reentrant libraries
- Ability to mix memory models with "near" and "far" pointers
- C and PL/M calling conventions for compatibility with PL/M and other C programs
- iRMX® interface libraries included

BUILT-IN FUNCTIONS

The iC-86/286 compiler features more than 35 processor-specific functions that directly generate machine code within the C language.

Built-in functions eliminate the need for in-line assembly language coding or making calls to assembly functions. This increases code performance and reduces programming time.

With built-ins you can enable or disable interrupts and directly control hardware I/O without having to exit C for assembler. This means you can write high performance software for real time applications without having to keep track of every architectural detail, as you would in assembly language. For example, to generate an INT instruction, you simply type:

```
causeinterrupt (number)
```

Or, the following iC-86 instruction will cause the processor to come to a halt with interrupts enabled:

```
halt( )
```

EMBEDDED COMPONENT SUPPORT

iC-86/286 compiler was designed specifically for embedded microprocessor applications. It produces ROMable code which can be loaded directly into target systems via Intel ICE emulators and debugged *without modification* for fast, easy, development and debugging.

HIGHLY OPTIMIZED

The iC-86/286 compiler has four levels of optimization for tailoring performance to your application. Important optimization features include a jump optimizer and improved register manipulation using register history.

RUN-TIME SUPPORT

Run-time libraries for the iC-86/286 compiler are designed for use in many environments. Both DOS and iRMX interface libraries are included so programs executing on those systems can take advantage of operating system features. The interface libraries conform to the ANSI standard. They also meet the IEEE standard POSIX interface so you can easily retarget the libraries for use in applications that do not run on DOS or iRMX.

The libraries are completely ROMable and re-entrant making it easy to adapt them for embedded, multi-tasking or real-time applications.

Both the DOS and iRMX-I operating system interface libraries are provided with iC-86 hosted on DOS. The iRMX-I hosted version of iC-86 includes the iRMX interface libraries only.

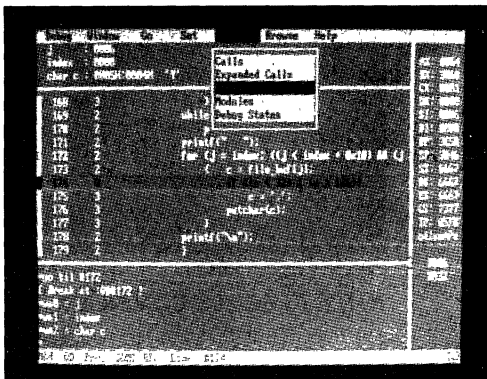
iC-286 compiler includes iRMX-II interface libraries only.

INTEGRATED DEBUG TOOLS

The iC-86/286 compiler is part of a completely integrated set of development tools from Intel (Fig. 1).

Code output from the compiler can be easily linked with modules written in assembler and high-level languages, such as PL/M, Fortran, and Pascal.

Linked modules and programs can be debugged using Intel's DB86 windowed source-level software debugger. The debugger uses an advanced interface with windows and pull-down menus for the ultimate in debug productivity. Watch windows can be opened to observe changing program variables and processor registers. You can readily switch between program modules and view the calling sequence and call stack.



Intel's DB86 Software Debugger

Naturally code generated by iC-86/286 works completely with Intel's iPICE, ICE-186, ICE-286 and ICE-386 family of in-circuit emulators as well as the iPAT™ performance analysis tool. This complete set of tools gives you the power to quickly debug, test, integrate and optimize your application code for the target system.



ICE-186, iPAT

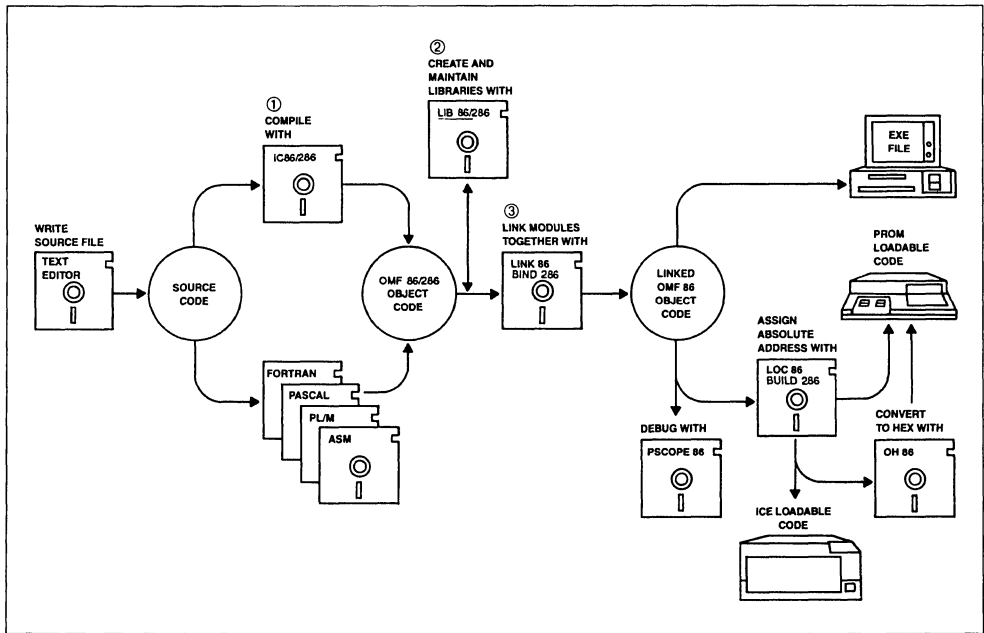


Figure 1: The Application Development Process

SERVICE AND SUPPORT

Intel's development tools are backed by our worldwide service and support organization dedicated to solving any problems encountered by our customers. Several hardware and software service programs are available

which include hotline support, consulting, training, technical newsletters, bulletin boards and other services. The iC-86/286 compiler includes 90 days of software support under warranty.

SPECIFICATIONS

ENVIRONMENT

Hardware Requirements	DOS Version:	IBM PC XT or AT (or 100% Compatible) running DOS 3.1 or greater.
	iRMX Version:	iRMX-I system for iC-86 iRMX-II system for iC-286
Memory Requirements	DOS Version:	256 KB
	iRMX Version:	374 KB
Media	DOS Version:	5 1/4" DS/DD Diskettes 3 1/2" DS/DD Diskettes
	iRMX Version:	DS/DD iRMX Standard Format

STANDARDS

iC-86/286 conforms to the X3J11 ANSI draft proposal for the C programming language.

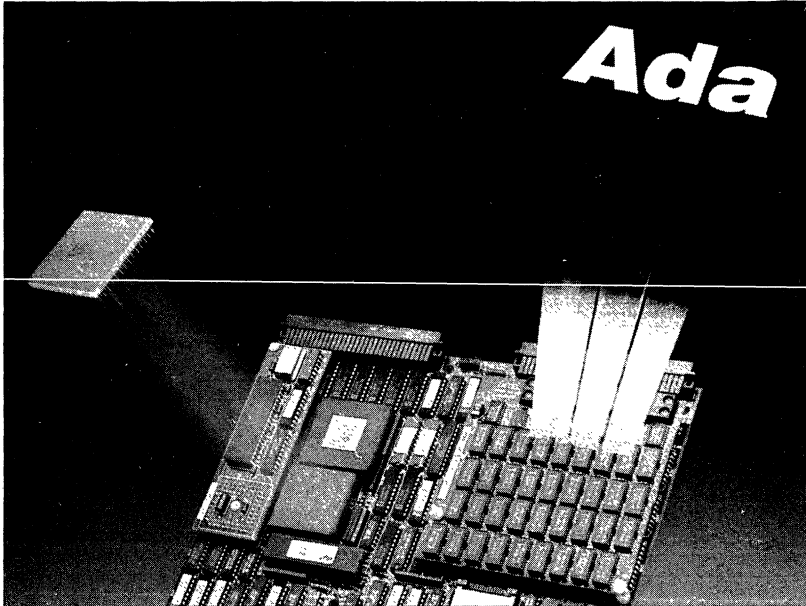
ORDERING INFORMATION

Order Code	Host Environment	Target Code
D86C86NL	DOS	8086/186
D86C286NL	DOS	80286
R86C86	iRMX-I	8086/186
R286C286	iRMX-II	80286

Other programming tools:

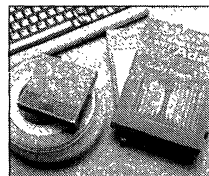
Order Code	Host Environment	Description
D86PAK86NL	DOS	8086/186 Assembler, DB86 Debugger, Utilities
D86ASM86KIT	DOS	AEDIT text editor 8086/186 Assembler, DB86 Debugger, Utilities
D86ASM286NL	DOS	80286 Assembler
R86ASM86	iRMX-I	8086/186 Assembler, Utilities
R286ASM286	iRMX-II	80286 Assembler, Utilities
RMXIIISFTSCP	iRMX-II	80286 Softscope Debugger

Ada: CROSS-DEVELOPMENT FOR THE 80386 MICROPROCESSOR



Intel's Ada*-386 Cross-Compilation Package comprises a rich set of Ada language tools for the programmer wanting to develop Ada applications targeted to the industry's leading 32-bit architecture, the 80386 microprocessor. This tool set includes an Ada cross compiler which generates compact, highly optimized, code for embedded real-time 80386 applications. The cross compiler and other tools making up the Cross-Compilation Package form a complete development environment designed specifically to support large scale, mission critical Ada programming development projects.

Intel's Ada-386 Cross-Compilation Package runs under VAX/VMS, and features, in addition to the cross compiler, a number of tools which make the programmer's job more efficient. These tools include a VMS hosted and targeted compiler to enable the programmer to do unit testing on the VAX early in the development cycle. Also included with each of the compilers is an Ada symbolic debugger to facilitate the debug process, and sophisticated code generation tools, such as the Linker and Global Optimizer, to help make the target code smaller and more efficient. An object module Importer is included with the cross compiler to allow the programmer to save and make use of program modules written in other Intel 80386 languages.



intel

KEY COMPILATION PACKAGE FEATURES

- Tight, efficient, 32-bit 80386 code
 - Generated code designed and optimized for the 80386 architecture
- Built-in support for the 80386
 - Full representation specifications, including address clauses
 - Machine code insertion
- 80387 coprocessor support
 - Full IEEE numerics support
- pragma INTERFACE
 - Call modules written in other Intel languages: ASM-386, PL/M-386 & C-386
- Highly optimized interrupt handling
 - Fast execution of interrupt handlers without requiring a context switch
- Pre-emptive delay
 - Force synchronization at the end of programmed "delays"
- Optional download and debug paths using the VAX-hosted Ada debugger
 - With a ROM-resident target debug monitor (included), or
 - ICE-386 (80386 In-Circuit Emulator) for less intrusive debugging
- Modular, configurable runtime system
 - Linker excludes routines not required by the embedded application (no overhead penalty)
 - Easily configured for different hardware environments

REAL TIME Ada FROM INTEL

Intel's Ada development environment makes developing real-time embedded applications convenient and easy. All steps in the development process can proceed, start to finish, from a VAX terminal—from initial unit testing with the VMS-targeted compiler to compiling and linking using the 80386-targeted cross compiler. Downloading to the 80386 target and debugging can also be accomplished from the VAX terminal.

COMPILATION PACKAGE COMPONENTS

- **Compilers & Library Tools**

Both compilers, the VMS targeted version and the 80386 cross targeted version, use the same user interface, commands and library management tools so the programmer learns them only once. The cross compiler has an optional switch which directs the compiler to produce assembly language text interspersed with Ada source text as comments. This feature gives the programmer a convenient way to hand-inspect the code. The assembly language text can also be assembled using the Intel 80386 Assembler.

The cross compiler has important optimizations to help meet real-time needs. For example, when response times to interrupts are critical, the

programmer can speed up response times by invoking the "function mapped" optimization via a special compiler directive, "pragma INTERRUPT." This function mapping enables an interrupt handler to execute without first requiring a task switch, i.e., within the context of the interrupted task.

- **Global Optimizer**

The Global Optimizer is used to reduce the size and increase the speed of embedded application code. This tool is invoked at the user's option, but usually after most of the coding and debugging is complete. Some key functions performed by the Global Optimizer include:

- **Linker**

The Linker combines separately compiled Ada modules, imported non-Ada modules (see Importer below), and the Ada runtime system into one executable image. To reduce target code size, the Linker also eliminates subprograms in the application code and in the runtime system that are not actually required by the application. The programmer may also use the Linker to produce output in a format suitable for burning PROMs.

- **Importer**

The Importer can help preserve previous software investments. The Importer converts object modules from Intel's OMF-386 format to a format suitable for linking with Ada modules. An Ada application can call these imported non-Ada modules through "pragma INTERFACE." Pragma INTERFACE is supported for Intel's ASM-386, PL/M-386 and C-386 languages.

- **Ada Runtime System**

All the necessary low-level support routines for executing programs on a bare 80386 microprocessor are provided in the Ada Runtime System. These routines are responsible for managing tasking, interrupts, the real-time clock and memory. Also included are predefined Ada packages, such as Text___I/O, IO___Exceptions, Unchecked___Conversion and Calendar. The Ada Runtime System is written almost entirely in Ada, with a small number of packages written in 80386 assembly language to support key low-level functions. The Runtime System is easily configured for different 80386 hardware environments, and source code is provided for this purpose.

• **Symbolic Debugger**

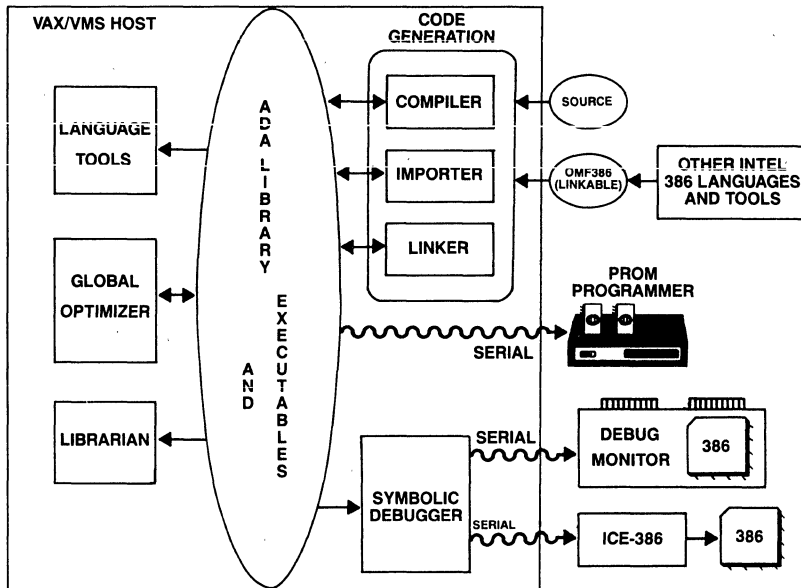
A VAX-resident symbolic debugger is supplied for each compiler. The debugger allows the programmer to debug at the source level while the code is executing on the target. Log and script files may be used to automate repetitive debug sessions. Important debugger features include:

Feature	Benefit
machine level interface	step through machine instructions; read/write to registers, memory, and I/O ports
single/multiple step	step through source code by single or multiple statements
breakpoints	halt execution at specified points
call chain display	display the dynamic nesting of a program at a particular point in time
task status display	determine the status of a task at a particular point in time
variable display	examine values of program variables
trap unhandled exceptions	examine state of the target program when an unhandled exception occurs

A small debug monitor, supplied in PROM, is used with the Symbolic Debugger. The code can also be downloaded and debugged using Intel's ICE-386 in-circuit emulator.

• **Language Tools**

Intel's Ada development environment includes tools that help development projects run more smoothly. A *Cross Referencer* provides a cross-reference listing of all source file locations where a user-defined symbol in an Ada compilation unit is defined. A *Source Dependency Lister* produces a valid compilation order list for compilation units in an Ada program and lists dependencies among units. A *Source Formatter* (or "pretty printer") takes as input a non-formatted Ada source file and outputs a formatted version of the same text that adheres to standardized language conventions.



WORLDWIDE SERVICE AND SUPPORT

Complete hardware and software support is provided. The Ada-386 Compilation Package comes with Intel's standard 90-day warranty plus an extended one-year maintenance agreement, ensuring uninterrupted support for a full 15 months. One-year maintenance contract renewals are available from Intel annually.

ORDERING INFORMATION

Order Code	Host Configuration	Product
VVSAda386-75	VAX 730, 750	Ada-386 Cross-Compilation Package. Included are the following tools hosted on VAX/VMS: <ul style="list-style-type: none">• 80386-targeted Ada cross compiler & library tools• VMS self-targeted Ada compiler & library tools• For each compiler a Symbolic Debugger, a Global Optimizer and Language Tools• Ada-386 Linker and Importer• Ada-386 Runtime System The Compilation Package also includes full documentation, Intel's standard 90-day warranty, and an extended one-year maintenance contract.
-82	VAX 78X, 8200	
-83	VAX 8300	
-85	VAX 85XX, 86XX, 8700	
-88	VAX 8800	
MVSAda386-VS	VAXStationII	80386 In-Circuit Emulator (hardware only). Used with the VAX-hosted Symbolic Debugger, the ICE-386 offers the programmer an alternative download method to using the ROM-resident debug kernel supplied with the Compilation Package.
-02	MicroVAXII	
ICE386HW	N/A	

Note: Ada-386 software license required for each host CPU. Multiple copies require multiple licenses.

*VAX/VMS is a registered trademark of Digital Equipment Corporation



COMPREHENSIVE DEVELOPMENT SUPPORT FOR THE INTEL386™ FAMILY OF MICROPROCESSORS

The perfect complement to the Intel386™ Family of microprocessors is the optimum development solution. From a single source, Intel, comes a complete, synergistic hardware and software development toolset, delivering full access to the power of the Intel386 architecture in a way that only Intel can.

Intel development tools are easy to use, yet powerful, with contemporary user interface techniques and productivity boosting features such as symbolic debugging. And you'll find Intel first to market with the tools needed to start development, and with lasting product quality and comprehensive support to keep development on-track.

If what interests you is getting the best product to market in as little time as possible, Intel is the choice.

FEATURES

- Comprehensive support for the full 32 bit Intel386 architecture, including protected mode and 4 gigabyte physical memory addressing
- Source display and symbolics allow debugging in the context of the original program
- Architectural extensions in Intel high-level languages provides for manipulating hardware directly without assembly language routines
- A common object code format (OMF-386™) supports the intermixing of modules written in various languages
- ROM-able code is output directly from the language tools, significantly reducing the effort necessary to integrate software into the final target system
- Support for the 80387 numeric coprocessor
- Operation in DOS and VAX*/VMS* environments

intel

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supersedes previously published specifications on these devices from Intel and is subject to change without notice.

© Intel Corporation 1989

February, 1989
Order Number: 280808-002

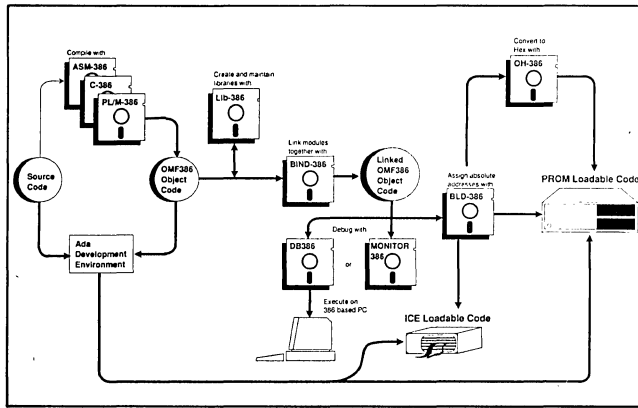


Figure 1. Intel Microprocessor Development Environment

ASM-386™ MACRO ASSEMBLER

ASM-386™ is a “high-level” macro assembler for the Intel386 Family. ASM-386 offers many features normally found only in high-level languages. The macro facility in ASM-386 saves development time by allowing common program sequences to be coded only once. The assembly language is strongly typed, performing extensive checks on the usage of variables and labels.

Other ASM-386 features include:

- “High-level” assembler mnemonics to simplify the language
- Structures and records for data representation
- Upward compatibility with ASM-286

PL/M-386™ COMPILER

PL/M-386™ is a structured high-level system implementation language for the Intel386 Family. PL/M-386 supports the implementation of protected operating system software by providing built-in procedures and variables to access the Intel386 architecture.

For efficient code generation, PL/M-386 features four levels of optimization, a virtual symbol table, and four models of program size and memory usage.

Other PL/M-386 features include:

- The ability to define a procedure as an interrupt handler as well as facilities for generating interrupts
- Direct support of byte, half-word, and word input and output from microprocessor ports
- Upward compatibility with PL/M-286 and PL/M-86 source code

PL/M-386 combines the benefits of a high-level language with the ability to access the Intel386 architecture. For the development of systems software, PL/M-386 is a cost-effective alternative to assembly language programming.

C-386™ COMPILER

C-386™ brings the C language to the Intel386 Family. For code efficiency, C-386 features two levels of optimization, three models of program size and memory usage, and an extremely efficient register allocator. The C-386 compiler eliminates common code, eliminates redundant loads and stores, and resolves span dependencies (shortens branches) within a program.

C-386 allows full access to the Intel386 architecture through control of bit fields, pointers, addresses, and register allocations.

Other C-386 features include:

- An interrupt directive defining a function as an interrupt function
- Built-in functions allow direct access to the microprocessor through the inline insertion of machine code
- Structure assignments, functions taking structure arguments, and returning structures, and the void and enum data types

The C-386 runtime library is implemented in layers. The upper layers include the standard I/O library (STDIO), memory management routines, conversion routines, and string manipulation routines. The lowest layer, operating system interface routines, is documented for adaptation to the target environment.

```

P      : 5B1F973C
mp     : -22
lines  : -2057400008

205  hboot(ptr);
206  stdout(" ");
207
208  * = ptr;
209  for (i=0;i<16;i++)
210  {
211  hboot(*ptr);
212
213  }
214  stdout(" ");
215  for (i=0;i<16;i++)
216  {
EAX 03F04138
EDX 00000000
ECX 00000001
EDI 00000000
ESI 00000000
ESP 00000040
EBP 000A0A25
EIP 00004500
IOP 00004500
FS 00000000
GS 00000000
CS 00004500
ZIP 000A0A20
od isa2c

*** int
structure
hpa +0
hl leng 970A0000
lines 970A0A92
[Debuging] Mod: MENU Proc: hdump Line: #197

```

RLL-386™ RELOCATION, LINKAGE, AND LIBRARY TOOLS

The RLL-386™ relocation, linkage, and library tools are a cohesive set of utilities featuring comprehensive support of the full Intel386 architecture.

RLL-386 provides for a variety of functions—from linking separate modules, building an object library, or linking in 80387 support, to building a task to execute under protected mode or the multi-tasking, memory protected system software itself.

The RLL-386 relocation, linkage, and library tools package includes a program binder for linking ASM, PL/M, and C modules together, a system builder for configuring protected, multi-task systems, a cross reference mapper, a program librarian, an 80387 numeric coprocessor support library, and a conversion utility for outputting hex format code for PROM programming.

Ada-386™ CROSS COMPILATION PACKAGE

The Ada-386™ Cross-Compilation Package is a complete development environment for embedded real-time Ada applications for the 386™ microprocessor. The Ada cross compiler, which runs under VAX/VMS, generates code highly optimized for the 80386. The Ada-386 Cross-Compilation Package also features a VMS hosted and targeted compiler and tools to support software debugging before the target system is available. Sophisticated code generation tools, such as the Global Optimizer, help make the target code smaller and more efficient.

Ada-386 includes a source level symbolic debugger working in unison with a small debug monitor supplied in PROM. Code can also be downloaded and debugged using Intel's ICE™-386 In-Circuit Emulator.

Other Ada-386 features include:

- The ability to directly call Intel's iRMK real-time kernel
- An object module importer allows program modules written in other Intel386 Family languages to be linked with Ada modules
- Built in support for the 386, including machine code insertion and full representation specifications
- Highly optimized interrupt handling—fast execution of interrupt handlers without requiring a context switch

INTEL386™ FAMILY IN-CIRCUIT EMULATORS

Intel386 Family in-circuit emulators embody exclusive technology that gives the emulator access to internal processor states that are accessible in no other way. Intel386 microprocessors fetch and execute instructions in parallel; fetched instructions are not necessarily executed. Because of this, an emulator without this access to internal processor

states is prone to error in determining what actually occurred inside the microprocessor. With Intel's exclusive technology, Intel386 Family emulators are one hundred percent accurate.

Other features of Intel386 Family in-circuit emulators include:

- Unparalleled support of the Intel386 architecture, notably the native protected mode
- Emulation at clock speeds to 25MHz and full-featured trigger and trace capabilities
- Non-intrusive operation
- Convertible to support any Intel386 microprocessor

With symbolic debugging, memory locations can be examined or modified using symbolic references to the original program, such as a procedure or a variable name, line number, or program label. Source code associated with a given line number can be displayed, as can the type information of variables, such as byte, word, record, or array. Microprocessor data structures, such as registers, descriptor tables, and page tables, can also be examined and modified using symbolic names. The symbolic debugging information for use with Intel development tools is produced only by Intel languages.

DOS-RESIDENT SOFTWARE DEBUGGER

DB386™ is an on-host software execution environment with source-level symbolic debug capabilities for object modules produced by Intel's ASM-386, iC-386, and PL/M-386 translators. This software debug environment allows 386 code to be executed and debugged directly on a 386 or 386SX™ microprocessor based PC, without any additional target hardware required. With Intel's standard windowed human interface, users can focus their efforts on finding bugs rather than spending time learning and manipulating the debug environment.

- **Supports key features of the 386™ architecture.** A run-time interface allows protected-mode 386 programs to be executed directly on a 386- or 386SX-based PC.
- **Ease of learning.** Drop-down menus make the tool easy to learn for new or casual users. A command line interface is also provided for more complex problems.
- **Extensive debug modes.** Watch windows (display user-specified variables), trace points, and breakpoints (including fixed, temporary, and conditional) can be set and modified as needed, even during a debug session.
- **See into the 386 application.** The user can browse source and call stacks, observe processor registers, and access watch window variables by either the pull down menu or by a single keystroke using the function keys. An easy-to-use disassembler and single-line assembler can also speed the debug process.

DOS-RESIDENT SOFTWARE DEBUGGER (continued)

- **Full debug symbolics for maximum productivity.** The user need not know whether a variable is an unsigned integer, a real, or a structure. The debugger utilizes the wealth of typing information available in Intel languages to display program variables in their respective type formats.

MONITOR-386™ SOFTWARE DEBUGGER

MONITOR-386™ is a software debugger primarily for non-PC 386 and 386SX microprocessor based target systems. The monitor allows 386 software applications to be downloaded and symbolically debugged on virtually any target system using the 386 architecture. MONITOR-386, used in conjunction with Intel single board computers iSBC® 386/22 and iSBC 386/116, can debug software before a functional prototype of the target system is available.

- **Breakpoints.** Both hardware and software breakpoints can be set at symbolic addresses.
- **Program Execution.** Users can single-step through assembly or high-level language applications.
- **Debug Procedures.** MONITOR-386 command sequences can be defined as macros, significantly reducing the amount of repetitive information which needs to be entered.
- **Disassembler/Single Line Assembler.** Users can display and patch memory with 80386/80387 mnemonics.

iPAT-386™ PERFORMANCE ANALYSIS TOOL

iPAT-386™ performance analysis tool provides analysis of real-time software executing on a 386-based target system. With iPAT-386, it is possible to speed-tune applications, optimize use of operating systems, determine response characteristics, and identify code execution coverage.

By examining iPAT-386 histogram and tabular information about procedure usage (with the option of including interaction with other procedures, hardware, the operating system, or interrupt service routines) for critical functions, performance bottlenecks can be identified. With iPAT-386 code execution coverage information, the completeness of testing can be confirmed. iPAT-386 can be used in conjunction with Intel's ICE-386™ in-circuit emulator to control test conditions.

iPAT-386 provides real-time analysis up to 20MHz, performance profiles of up to 125 partitions, and code execution coverage analysis over 252K.

Intel386, 386, 386SX, 376, ICE, and iSBC are trademarks of Intel Corporation. VAX and VMS are registered trademarks of Digital Equipment Corporation.

SERVICE, SUPPORT, AND TRAINING

To augment its development tools, Intel offers a full array of seminars, classes, and workshops, field application engineering expertise, hotline technical support, and on-site service.

PRODUCT SUPPORT MATRIX

Product	Component			Host	
	386	386sx	376	DOS	VAX/VMS
ASM-386 Macro Assembler	✓	✓	✓	✓	✓
PLM-386 Compiler	✓	✓	✓	✓	✓
C-386 Compiler	✓	✓	✓	✓	✓
RL-386 Relocation, Linkage, and Library tools	✓	✓	✓	✓	✓
Ada-386 Cross Compilation Package	✓				✓
Intel386 Family In-Circuit Emulators	✓	✓	✓	✓	
Windowed Software Debugger	✓	✓		✓	
Monitor-386 Software Debugger	✓	✓		✓	
iPAT-386 Performance Analysis Tool	✓			✓	

ORDERING INFORMATION

For direct information on Intel's Development Tools, or for the number of your nearest sales office or distributor, call 800-874-6835 (U.S.). For information or literature on additional Intel products, call 800-548-4725 (U.S. and Canada).

UNITED STATES, Intel Corporation
3065 Bowers Ave., Santa Clara, CA 95051
Tel: (408) 765-8080

JAPAN, Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi,
Ibaraki, 300-26
Tel: 029747-8511

FRANCE, Intel Corporation S.A.R.L.
1, Rue Edison, BP 303,
78054 Saint-Quentin-en-Yvelines Cedex
Tel: (33) 1-30 57 70 00

UNITED KINGDOM, Intel Corporation (U.K.) Ltd.
Pipers Way, Swindon, Wiltshire, England SN3 1RJ
Tel: (0793) 696000

WEST GERMANY, Intel Semiconductor GmbH
Dornacher Strasse 1, 8016 Feldkirchen bei Muenchen
Tel: 089/90 99 20

HONG KONG, Intel Semiconductor Ltd.
10/F East Tower, Bond Center,
Queensway, Central
Tel: (5) 8444-555

CANADA, Intel Semiconductor of Canada, Ltd.
190 Attwell Drive, Suite 500
Rexdale, Ontario M9W 6H8
Tel. (416) 675-2105

AEDIT SOURCE CODE AND TEXT EDITOR



PROGRAMMER SUPPORT

AEDIT is a full-screen text editing system designed specifically for software engineers and technical writers. With the facilities for automatic program block indentation, HEX display and input, and full macro support, AEDIT is an essential tool for any programming environment. And with AEDIT, the output file is the pure ASCII text (or HEX code) you input—no special characters or proprietary formats.

Dual file editing means you can create source code and its supporting documents at the same time. Keep your program listing with its errors in the background for easy reference while correcting the source in the foreground. Using the split-screen windowing capability, it is easy to compare two files, or copy text from one to the other. The DOS system-escape command eliminates the need to leave the editor to compile a program, get a directory listing, or execute any other program executable at the DOS system level.

There are no limits placed on the size of the file or the length of the lines processed with AEDIT. It even has a batch mode for those times when you need to make automatic string substitutions or insertions in a number of separate text files.

AEDIT FEATURES

- Complete range of editing support—from document processing to HEX code entry and modification
- Supports system escape for quick execution of PC-DOS System level commands
- Full macro support for complex or repetitive editing tasks
- Hosted on PC-DOS and RMX operating systems
- Dual file support with optional split-screen windowing
- No limit to file size or line length
- Quick response with an easy to use menu driven interface
- Configurable and extensible for complete control of the editing process

FEATURES

POWERFUL TEXT EDITOR

As a text editor, AEDIT is versatile and complete. In addition to simple character insertion and cursor positioning commands, AEDIT supports a number of text block processing commands. Using these commands you can easily move, copy, or delete both small and large blocks of text. AEDIT also provides facilities for forward or reverse string searches, string replacement and query replace.

AEDIT removes the restriction of only inserting characters when adding or modifying text. When adding text with AEDIT you may choose to either insert characters at the current cursor location, or over-write the existing text as you type. This flexibility simplifies the creation and editing of tables and charts.

USER INTERFACE

The menu-driven interface AEDIT provides makes it unnecessary to memorize long lists of commands and their syntax. Instead, a complete list of the commands or options available at any point is always displayed at the bottom of the screen. This makes AEDIT both easy to learn and easy to use.

FULL FLEXIBILITY

In addition to the standard PC terminal support provided with AEDIT, you are able to configure AEDIT to work with almost any terminal. This along with user-definable macros and full adjustable tabs, margins, and case sensitivity combine to make AEDIT one of the most flexible editors available today.

MACRO SUPPORT

AEDIT will create macros by simply keeping track of the command and text that you type, "learning" the function the macro is to perform. The editor remembers your actions for later execution, or you may store them in a file to use in a later editing session.

Alternatively, you can design a macro using AEDIT's powerful macro language. Included with the editor is an extensive library of useful macros which you may use or modify to meet your individual editing needs.

TEXT PROCESSING

For your documentation needs, paragraph filling or justification simplifies the chore of document formatting. Automatic carriage return insertion means you can focus on the content of what you are typing instead of how close you are to the edge of the screen.

SERVICE, SUPPORT, AND TRAINING

Intel augments its development tools with a full array of seminars, classes, and workshops; on-site consulting services; field application engineering expertise; telephone hot-line support; and software and hardware maintenance contracts. This full line of services will ensure your design success.

SPECIFICATIONS

HOST SYSTEM

AEDIT for PC-DOS has been designed to run on the IBM* PC XT, IBM PC AT, and compatibles. It has been tested and evaluated for the PC-DOS 3.0 or greater operating system.

Versions of AEDIT are available for the iRMX™-86 and RMX II Operating System.

ORDERING INFORMATION

D86EDINL	AEDIT Source Code Editor Release 2.2 for PC-DOS with supporting documentation
122716	AEDIT-DOS Users Guide
122721	AEDIT-DOS Pocket Reference
RMX864WSU	AEDIT for iRMX-86 Operating System
R286EDI286EU	AEDIT for iRMX II Operating System

For direct information on Intel's Development Tools, or for the number of your nearest sales office or distributor, call 800-874-6835 (U.S.). For information or literature on additional Intel products, call 800-548-4725 (U.S. and Canada).

iPAT™ PERFORMANCE ANALYSIS TOOL



REAL-TIME SOFTWARE ANALYSIS FOR THE 8086/88, 80186/188, 80286, AND 80386

Intel's iPAT™ Performance Analysis Tool enables OEMs developing applications based on the 8086/88, 80186/188, 80286, or 80386 microprocessors to analyze real-time software execution in their prototype systems at speeds up to 20 MHz. Through such analysis, it is possible to speed-tune applications with real-time data, optimize use of operating systems (such as Intel's iRMX® II Real-Time Multitasking Executive for the 80286 and 80386, and iRMK™ Real-Time Multitasking Kernel for the 80386), characterize response characteristics, and determine code execution coverage by real-time test suites. Analysis is performed symbolically, non-intrusively, and in real-time with 100% sampling in the microprocessor prototype environment. iPAT supports analysis of OEM-developed software built using 8086, 80286, and 80386 assemblers and compilers supplied by Intel and other vendors.

All iPAT Performance Analysis Tool products are serially linked to DOS computer systems (such as IBM® PC AT, PC XT, and PS/2® Model 80) to host iPAT control and graphic display software. Several means of access to the user's prototype microprocessor system are supported. For the 80286 (real and protected mode), a 12.5 MHz iPAT-286 probe can be used with the iPATCORE system. For the 8086/88 (MAX MODE designs only), a 10 MHz iPAT-88 probe can be used with the iPATCORE system. iPATCORE systems also can be connected to sockets provided on the ICE™-286 and ICE-186 in-circuit emulators, or interfaced to IICE in-circuit emulators with probes supporting the 8086/88, 80186/188, or 80286. The 20 MHz iPAT™-386™ probe, also supported by the common iPATCORE system, can be operated either in "piggyback" fashion connected to an Intel ICE in-circuit emulator for the Intel386™, or directly connected to a prototype system independent of an ICE. iPAT-386 supports all models of 80386 applications anywhere in the lowest 16 Megabytes of the 80386 linear address space.

iPAT FEATURES

- Up to 20 MHz real-time analysis
- Histograms and analysis tables
- Performance profiles of up to 125 partitions
- Code execution coverage over up to 252K
- Hardware or software interrupt analysis
- Simple use with function keys and graphics
- Use with or without Intel ICES



FEATURES

MOST COMPLETE REAL-TIME ANALYSIS AVAILABLE TODAY

iPAT Performance Analysis Tools use in-circuit probes containing proprietary chip technology to achieve full sampling in real-time non-intrusively.

MEETS THE REAL-TIME DESIGNER'S NEEDS

The iPAT products include support for interactions between real-time software and hardware interrupts, real-time operating systems, "idle time," and full analysis of real-time process control systems.

SPEED-TUNING YOUR SOFTWARE

By examining iPAT histogram and tabular information about procedure usage (including or not including their interaction with other procedures, hardware, operating systems, or interrupt service routines) for critical functions, the software engineer can quickly pinpoint trouble spots. Armed with this information, bottlenecks can be eliminated by means such as changes to algorithms, recoding in assembler, or adjusting system interrupt priorities. Finally, iPAT can be used to prove the acceptability of the developer's results.

EFFICIENCY AND EFFECTIVENESS IN TESTING

With iPAT code execution coverage information, product evaluation with test suites can be performed more effectively and in less time. The evaluation team can quickly pinpoint areas of code that are executed or not executed under real-time conditions. By this means, the evaluation team can substantially remove the "black box" aspect of testing and assure 100% hits on the software under test. Coverage information can be used to document testing at the module, procedure, and line level. iPAT utilities also support generation of instruction-level code coverage information.

ANALYSIS WITH OR WITHOUT SYMBOLICS

If your application is developed with "debug" symbolics generated by Intel 8086, 80286, or 80386 assemblers and compilers, iPAT can use them — automatically. Symbolic names also can be defined within the iPAT environment, or conversion tools supplied with the iPAT products can be used to create symbolic information from virtually any vendor's map files for 8086, 80286, and 80386 software tools.

REAL OR PROTECTED MODE

iPAT supports 80286 and 80386 protected mode symbolic information generated by Intel 80286 and 80386 software tools. It can work with absolute addresses, as well as base-offset or selector-offset references to partitions in the prototype system's execution address space.

FROM ROM-LOADED TO OPERATING SYSTEM LOADED APPLICATIONS

The software analysis provided by iPAT watches absolute execution addresses in-circuit in real time, but also supports use of various iPAT utilities to determine the load locations for load-time located software, such as applications running under iRMXII, DOS, Microsoft Windows*, or MS*OS/2.

USE STANDALONE OR WITH ICE

The iPAT-386, iPAT-286, and iPAT-86/88 probes, together with an iPATCORE system, provide standalone software analysis independent of an ICE (in-circuit emulator) system. The iPATCORE system and DOS-hosted software also can be used together with ICE-386, ICE-286, and iICE-86/88, 186/188, or 286 in-circuit emulators and DOS-hosted software. Under the latter scenario, the user can examine prototype software characteristics in real-time on one DOS host while another DOS host is used to supply input or test conditions to the prototype through an ICE. It also is possible to use an iPATCORE and iICE system with integrated host software on a single Intel Series III or Series IV development system or on a DOS computer.

UTILITIES FOR YOUR NEEDS

Various utilities supplied with iPAT products support generation of symbolic information from map files associated with 3rd-party software tools, extended analysis of iPAT code execution coverage analysis data, and convenience in the working environment. For example, symbolics can be generated for maps produced by most software tools, instruction-level code execution information can be produced, and iRMXII-format disks can be read/written in DOS floppy drives to facilitate file transfer.

WORLDWIDE SERVICE AND SUPPORT

All iPAT Performance Analysis Tool products are supported by Intel's worldwide service and support. Total hardware and software support is available, including a hotline number when the need is there.

FEATURES

CONFIGURATION GUIDE

For all of the following application requirements, the iPAT system is supported with iPAT 2.0 (or greater) or iPAT/ICE 1.2 (or greater) host software, as footnoted.

Application Software	Option	iPAT Order Codes	Host System
80386 Embedded	#1	iPAT386DOS1, iPATCORE	DOS
iRMK on 80386	#1	iPAT386DOS, iPATCORE	DOS
iRMXII OS-Loaded or Embedded on 386	#1	iPAT386DOS, iPATCORE	DOS
OS/2-Loaded on 386	#1	iPAT386DOS, iPATCORE	DOS
iRMXII OS-Loaded or Embedded	#1	iPAT286DOS, iPATCORE	DOS
80286 Embedded	#1	iPAT286DOS, iPATCORE	DOS
	#2	ICEPATKIT2	DOS
	#3	ICEPATKIT3	DOS
	#4	IIIPATD, iPATCORE3	DOS4
	#5	IIIPATB, iPATCORE3	Series III4
	#6	IIIPATC, iPATCORE3	Series IV4
DOS OS-Loaded 80286	#1	iPAT286DOS, iPATCORE	DOS
OS/2 OS-Loaded 80286	#1	iPAT286DOS, iPATCORE	DOS
80186/188 Embedded	#1	ICEPATKIT2	DOS
	#2	ICEPATKIT3	DOS
	#3	IIIPATD, iPATCORE3	DOS4
	#4	IIIPATB, iPATCORE3	Series III4
	#5	IIIPATC, iPATCORE3	Series IV4
DOS OS-Loaded 8086/88	#1	iPAT88DOS, iPATCORE	DOS
8086/88 Embedded	#1	iPAT88DOS, iPATCORE	DOS
	#2	ICEPATKIT3	DOS
	#3	IIIPATD, iPATCORE3	DOS4
	#4	IIIPATB, iPATCORE3	Series III4
	#5	IIIPATC, iPATCORE3	Series IV4

Notes:

1. Operable standalone or with ICE-386 (separate product; separate host). iPAT-386 probe connects directly to prototype system socket, or to optional 4 probe-to-socket hinge cable (order code TA386A), or to ICE-386 probe socket.
2. Requires ICE-186 or ICE-286 in-circuit emulator system.
3. Requires ICE in-circuit emulator system.
4. Includes iPAT/ICE integrated software (iPAT/ICE 1.2 or greater), which only supports sequential iPAT and ICE operation on one host, rather than in parallel on two hosts (iPAT 2.0 or greater).

SPECIFICATIONS

HOST COMPUTER REQUIREMENTS

All iPAT Performance Analysis Tool products are hosted on IBM PC AT, PC XT, or PS/2 Model 80 personal computers, or 100% compatibles, and use a serial link for host-to-iPAT communications. At least a PC AT class system is recommended. The DOS host system must meet the following minimum requirements:

- 640K Bytes of Memory
- 360K Byte or 1.2M Byte floppy disk drive
- Fixed disk drive
- A serial port (COM1 or COM2) supporting 9600 baud data transfer
- DOS 3.0 or later
- IBM or 100% compatible BIOS

PHYSICAL DESCRIPTIONS

Unit	Width		Height		Length	
	Inches	Cm.	Inches	Cm.	Inches	Cm.
iPATCORE	8.25	21.0	1.75	4.5	13.75	35.0
Power Supply	7.75	20.0	4.25	11.0	11.0	28.0
iPAT-386 probe	3.0	7.6	0.50	1.3	4.0	10.1
iPAT-286 probe	4.0	10.2	1.12	2.8	6.0	15.3
iPAT-86 probe	4.0	10.2	1.12	2.8	6.0	15.3
iPATCABLE (to ICE-186/286)	4.0	10.2	.25	.6	36.0	91.4
IIIPATB,C,D (IICE board)	12.0	30.5	12.0	30.5	.5	1.3
Serial cables PC AT/XT PS/2					144.0	370.0

ELECTRICAL CONSIDERATIONS

The iPATCORE system power supply uses an AC power source at 100V, 120V, 220V, or 240V over 47Hz to 63Hz. 2 amps (AC) at 100V or 120V; 1 amp at 220V or 240V.

iPAT-386, iPAT-286 and iPAT-86/88 probes are externally powered, impose no power demands on the user's prototype, and can thus be used to analyze software activity through power down and power up of a prototype system. For ICE-386, ICE-286, ICE-186, and IICE microprocessor probes, see the appropriate in-circuit emulator factsheets.

ENVIRONMENTAL SPECIFICATIONS

Operating Temperature: 10°C to 40°C (50°F to 104°F) ambient

Operating Humidity: Maximum of 85% relative humidity, non-condensing



TARGET: UNIX V/386 ADA APPLICATIONS

The UNIX* V.3/386 Ada toolset consists of an Ada compiler plus a set of related tools that allow a user to develop, manage, compile, optimize, link, load, and execute Ada application programs on an Intel386™ Microprocessor-based UNIX host. Supported hosts are Intel's MULTIBUS® II Development Platform: the MDP, and Intel386™ Microprocessor-based PC platforms: the 301 and 302 (plus 100% compatibles).

Both the compiler and the object code it produces run under the UNIX operating system, resulting in a user interface that is simple, consistent, and familiar to UNIX users. Intel's UNIX/386 Ada toolset provides a flexible, price effective, project-oriented development environment for developing commercial, industrial, and military Ada applications.

FEATURES:

- complete Ada development environment
- generated code is fast and efficient (over 5200 Dhrystones/second)
- sophisticated, Ada-knowledgeable GLOBAL OPTIMIZER
- ACVC validated configurations
- worldwide support provided by Intel



THIS PRODUCT CONFORMS
TO ANSI/MIL-STD-1815A AS
DETERMINED BY THE AJPO
UNDER ITS CURRENT
TESTING PROCEDURES

intel



TOOLSET COMPONENTS



UNIX-targeted Ada COMPILER

Compiles Ada source (at a rate of approximately 1000 Ada source lines per CPU minute on an unloaded Intel MDP) while performing syntax checking, semantic analysis, external calls validation, and error reporting. The resultant "intermediate form," along with optional debug information, is put into the project database. While processing source, the compiler automatically performs many "base level" optimizations, such as dead code elimination and register assignment for loops.

Intelligent LINKER

The intelligent LINKER binds and links the user's application with the Ada Execution Environment (AEE) to build an image that's ready for execution or debugging under UNIX. To reduce final code size, the intelligent LINKER automatically includes only those modules from the user's application and from the AEE that will actually be needed in the final configuration. The LINKER outputs object code in COFF format.

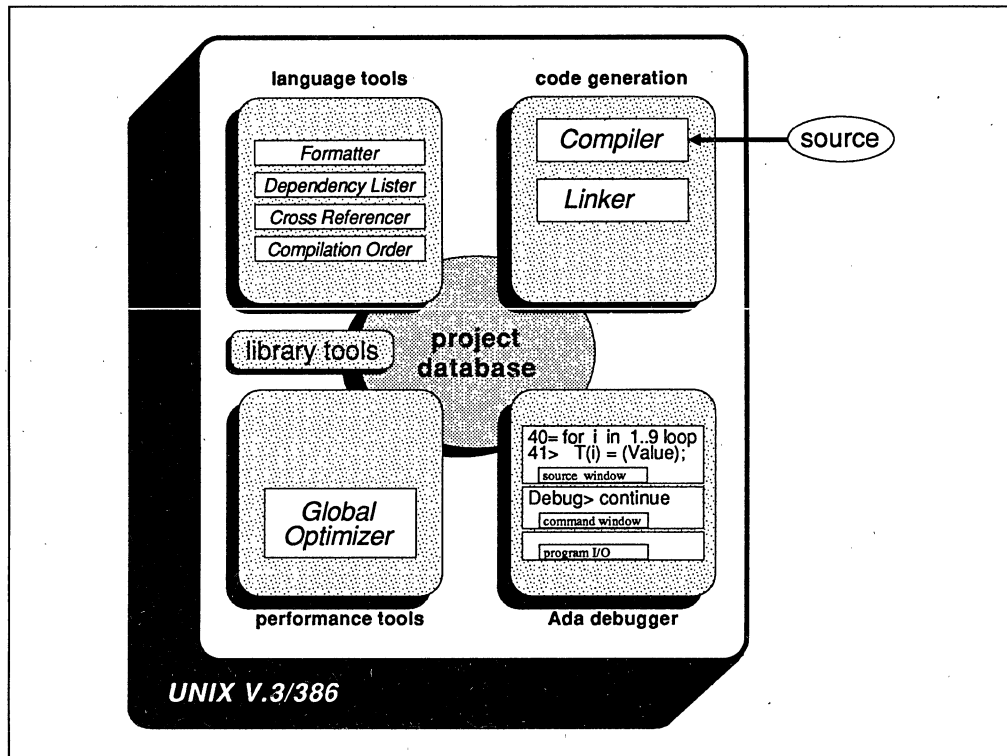
GLOBAL OPTIMIZER

The GLOBAL OPTIMIZER is a powerful tool for decreasing code size and increasing execution speed of the code generated by the COMPILER. The GLOBAL OPTIMIZER is able to optimize single compilation units, groups of compilation units (known as collections), or the entire executable image. As a general rule, Globally Optimized systems are up to 30% faster and 30% smaller than their unoptimized forms.

Source-level, Symbolic Ada DEBUGGER

The DEBUGGER provides an easy, clear, and interactive environment for examining the behavior of Ada programs during execution. The windowing feature of the DEBUGGER allows the user to split the debugger screen into two areas: a source window (where source can be viewed a page at a time), and a command line window. The DEBUGGER has extensive capabilities including:

- *breakpoint on execution*
- *source level single step*
- *call chain display*
- *task status display*
- *program I/O display*
- *macro processor*



AEE Ada Execution Environment (or “run-time system”)

The AEE provides the environment for executing Ada programs on the UNIX host. The following Ada packages are provided with the product:

- TEXT_IO
- SEQUENTIAL_IO
- DIRECT_IO
- UNCHECKED_CONVERSION
- CALENDAR

LANGUAGE TOOLS

The following Ada Language Tools supplement the standard UNIX toolset and provide the developer with specific Ada capabilities to improve programmer productivity.

- Ada Cross Referencer
generates a listing of all symbolic names referenced within a compilation unit or collection and shows where they were declared, used, and referenced

- Ada Source Dependency Lister
reports dependencies among units and/or produces a valid compilation order
- Ada Source Formatter (often referred to as a “pretty printer”)
formats Ada source text so that it’s easy to read and consistent in appearance
- Ada Compilation Order Tool
provides the user the ability to compile or re-compile systems with a single command

LIBRARY MANAGEMENT TOOLS

The LIBRARY MANAGER and LIBRARY TOOLSET are a powerful set of utilities for managing the Ada project database. The utilities work with the Ada database to provide:

- library creation/deletion and listing
- version control
- configuration management
- order of compilation rules
- elaboration ordering
- system builds

SYSTEM REQUIREMENTS

System requirements are:

- 4M-bytes RAM (bare minimum), 6M-bytes or more are recommended
- 60M-byte hard disk or larger (Ada toolset takes up 20M-bytes, UNIX takes up another 25M-bytes)
- 1/4" cartridge tape drive
- 80387 math co-processor is NOT required

UNIX is bundled with the Intel MDP and is available for the 301 and 302 PC platforms by contacting Intel at (800) FOR-UNIX. The Ada toolset is delivered on a single 1/4" cartridge tape ("tar" format).

VALIDATION

The UNIX V/386 Self-Targeted Ada compilation system was validated by the Ada Joint Program Office (AJPO) according to the following criteria:

Date of Issue: Sept. 29, 1988, ACVC V1.9

Certificate #: 88052911.09137

Host: Intel MDP, MULTIBUS II (UNIX, Version V3.0) (a derived validation was awarded for the 301)

Target: Same as host

ORDERING INFORMATION

	<i>platform</i>
U386AdaSWS	Intel's MB-II MDP
U386AdaSPS	Intel's 301/302's (plus 100% compatible 386 PCs)

The codes listed above include the Ada toolset plus 15-months of hot-line support and product upgrades.

	<i>platform</i>
U386AdaSW	Intel's MB-II MDP
U386AdaSP	Intel's 301/302's (plus 100% compatible 386 PCs)

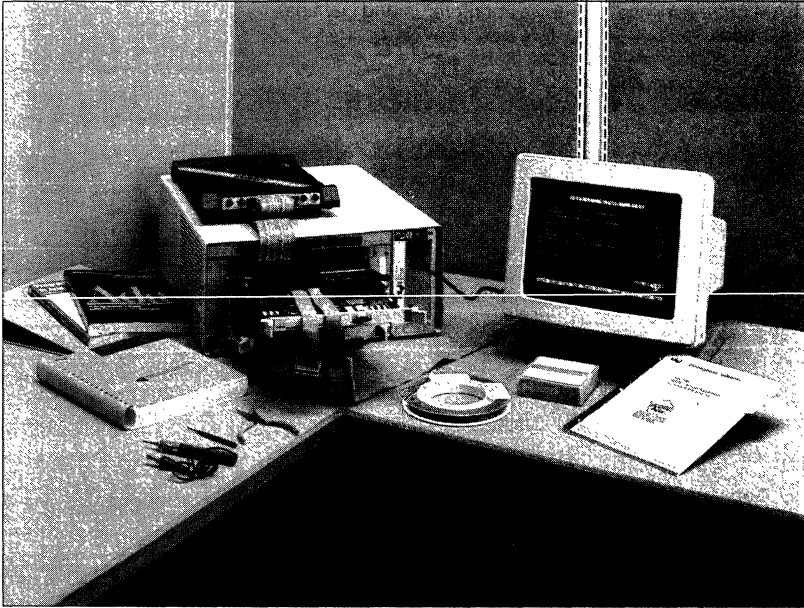
The codes listed above include the Ada toolset plus 3-months of hot-line support and product upgrades.

NOTE: Each copy of the Ada toolset is licensed for use on a single system. Multiple copies require multiple licenses.

WORLDWIDE SERVICE AND SUPPORT

The Ada toolset comes complete with either Intel's standard 90-day warranty, or an extended 15-month maintenance plan. Follow-on support plans are also available from Intel.

For more information or the number of your nearest Intel sales office, call 800-548-4725 (good in the U.S. and Canada).

**TARGET: REAL-TIME, EMBEDDED 386™ DESIGNS**

The *Ada-386/iRMK™ interface libraries* allow users of Release 1.1 of Intel's validated VAX/VMS* hosted Ada-386 toolset the ability to integrate iRMK's features and benefits into their embedded, real-time Intel386™ microprocessor based Ada applications.

Intel's iRMK™ is a small, fast, configurable, 32-bit kernel designed for real-time applications that use the 386™ microprocessor. iRMK provides a tasking system (controlled by an interrupt-driven, deterministic, priority-based scheduler), intertask communication and synchronization primitives, fast memory management, and MULTIBUS® II message passing primitives.

FEATURES:

- access to MULTIBUS® II resources through iRMK primitives (message passing, . . .),
- iRMK tasking as an alternative to Ada tasking,
- fast and flexible inter-iRMK-task communication (semaphores, mailboxes, . . .), and
- high-level access to iRMK supported hardware drivers (interrupt controllers, timers, . . .).

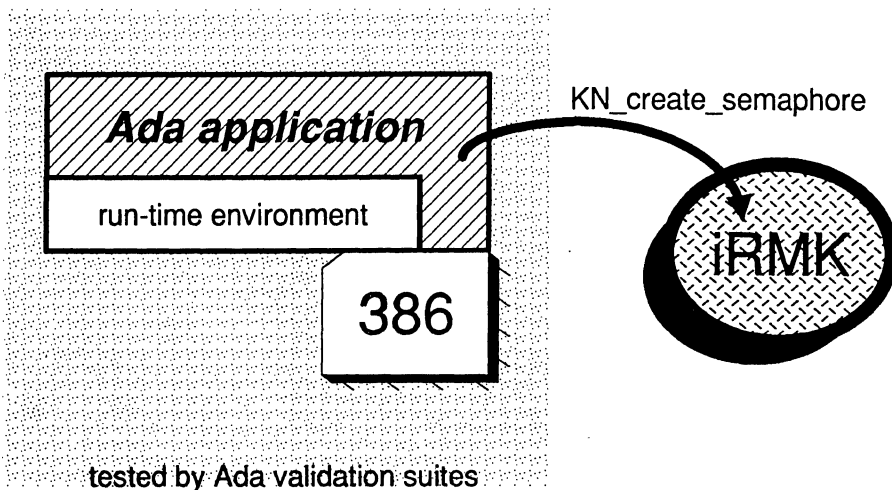


ACCESSING iRMK™ FROM ADA-386

All iRMK1.1 primitives are accessible from Ada-386 programs (except for *KN_initialize* which is automatically called from the Ada start-up code).

To use iRMK in your Ada-386 application, simply add the appropriate Ada sub-library to your library list and

use iRMK primitives as if they were Ada procedures or functions. Use Ada context clauses to identify the modules your application needs, just as you *with* other library units. Thus, accessing iRMK from Ada-386 is analogous to putting iRMK primitive calls in a C-386 or PL/M-386 program.



TASKING MODELS

Ada-386 programs can either use the iRMK tasking model or the Ada tasking model (but not both).

Using iRMK™ Tasking

By using the iRMK tasking model, you gain access to an environment that can provide:

- the ability to map Ada procedures and functions to iRMK tasks,
- bounded interrupt latency,
- fast and flexible inter-task communication,
- no run-time degradation with increasing tasks,
- fast memory allocation, and
- MULTIBUS® II message passing.

If iRMK tasking is chosen, Ada task specifications and task bodies cannot be present in the software (these would invoke the Ada tasking environment). Because Ada tasks cannot be used, the Ada keywords: *accept*, *select*, and *entry* are not available.

Using Ada Tasking

If Ada tasking is chosen, a subset of iRMK primitives can still safely be used. These include: iRMK's Device Management Module primitives, Memory Management Module primitives, Interconnect Space Support Module primitives, and Descriptor Table Management Module primitives.

In addition, the Ada *delay* function maps directly onto the iRMK *KN_sleep* primitive, providing deterministic task awoken times.

Other benefits provided are:

- management of Ada's heap space by iRMK's fast memory manager, and,
- a high-level interface to iRMK hardware drivers.

LIBRARY CONTENTS

The *Ada-386/iRMK interface libraries* consist of Release I.1 of the iRMK kernel, plus associated interface modules, that have been IMPORTed into Ada-386 sub-libraries. Two Ada sub-libraries are provided:

- RMK386:
OPTIMIZED for maximum performance with minimum size, and
- RMK_DBG:
un-OPTIMIZED and un-SQUEEZED to be used for debugging and collective optimization with your application

Each sub-library contains:

- the iRMK basic and optional modules (in object form), ready to be LINKed with your Ada-386 application,
- Ada specifications and bodies for the iRMK primitives,
- Ada-to-iRMK interface modules (in object form), and
- Ada-386 Run-Time Environment modules (in ASM-386 source form)
 - environment code for MULTIBUS® I and MULTIBUS II boards,
 - initialization code for MULTIBUS® I and MULTIBUS II boards, and
 - Ada *delay* and timer code.

Two Linker options files (configured for Intel's 16 and 20MHz 386/MB-I boards and Intel's 16 and 20MHz 386/MB-II boards) and three sample Ada-386 programs (in source form) are also included. The sample programs demonstrate:

- iRMK Tasking
ten tasks are created (using the *KN_create_task* primitive)
- Inter-Task Communication
parameters are passed among three iRMK tasks via a FIFO mailbox
- MULTIBUS II Message Passing
this example demonstrates board-to-board communication.

WORLDWIDE SERVICE AND SUPPORT

The Ada portion of the *Ada-386/iRMK interface libraries* is covered, at no extra charge, by the 15-month Ada-386 maintenance plan. However, specific support for the iRMK kernel is not covered by the Ada-386 plan. Support for iRMK is available from Intel under a separate policy. Contact your local sales office for further information.

ORDERING INFORMATION

order code	magnetic media
MVVSAdaRMKRI.1	TK50
VVSAAdaRMKRI.1	7", 1600BPI tape

Both products are shipped in VMS BACKUP format.

For more information or the number of your nearest Intel sales office, call 800-548-4725 (good in the U.S. and Canada).

i486™ MICROPROCESSOR DEVELOPMENT TOOLS**COMPREHENSIVE DEVELOPMENT SUPPORT FOR THE i486™ MICROPROCESSOR**

The perfect complement to the i486™ microprocessor is the optimum development solution. From a single source, Intel, comes a complete, synergistic hardware and software development toolset, delivering full access to the power of the i486 architecture in a way that only Intel can.

Intel development tools are easy to use, yet powerful, with contemporary user interface techniques and productivity boosting features such as symbolic debugging. And you'll find Intel first to market with the tools needed to start development, and with lasting product quality and comprehensive support to keep development on-track.

If what interests you is getting the best product to market in as little time as possible, Intel is the choice.

FEATURES

- Comprehensive support for the full 32 bit i486 architecture, including protected mode, 4 gigabyte physical memory addressing, and caching
- Symbolics allow debugging in the context of the original program
- Architectural extensions in Intel high-level languages provide for manipulating hardware directly without assembly language routines
- 386™ software compatibility is assured at both object and source level
- ROM-able code is output directly from the language tools, significantly reducing the effort necessary to integrate software into the final target system
- Support for the i486 microprocessor on-chip numerics with numerics libraries
- Operation in DOS and VAX*/VMS* environments

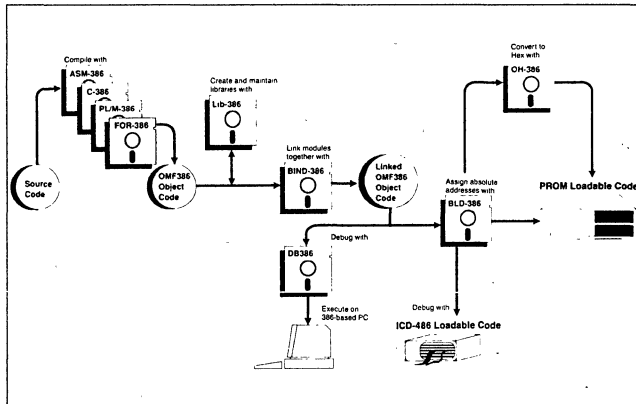


Figure 1. Intel i486™ Microprocessor Development Environment

ASM-386/486 MACRO ASSEMBLER

ASM-386/486 is a "high-level" macro assembler for the i486 microprocessor. ASM-386/486 offers many features normally found only in high-level languages. The macro facility in ASM-386/486 saves development time by allowing common program sequences to be coded only once. The assembly language is strongly typed, performing extensive checks on the usage of variables and labels.

Other ASM-386/486 features include:

- "High-level" assembler mnemonics to simplify the language
- Structures and records for data representation
- Support for Intel's standard object code format for symbolic debug, and for linking object modules from other Intel 386/486 languages.

PL/M-386/486 COMPILER

PL/M-386/486 is a structured high-level system implementation language for the i486 architecture. PL/M-386/486 supports the implementation of protected operating system software by providing built-in procedures and variables to access the i486 architecture.

For efficient code generation, PL/M-386/486 features four levels of optimization, a virtual symbol table, and four models of program size and memory usage.

Other PL/M-386/486 features include:

- The ability to define a procedure as an interrupt handler as well as facilities for generating interrupts
- Direct support of byte, half-word, and word input and output from microprocessor ports
- Support for Intel's standard object code format for symbolic debug, and for linking object modules from other Intel 386/486 languages.

PL/M-386/486 combines the benefits of a high-level language with the ability to access the i486 architecture. For the development of systems software, PL/M-386/486 is a cost-effective alternative to assembly language programming.

C-386/486 COMPILER

C-386/486 is ideal for developing portable system applications based on the i486 architecture. For code efficiency, C-386/486 utilizes two levels of optimization, three models of program size and memory usage, and an extremely efficient register allocation scheme. In addition, C-386/486 allows full access to the 486 architecture through control of bit fields, pointers, addresses, and register allocation.

The C-386/486 runtime library is implemented in layers. The upper layers include the standard I/O library (STDIO), memory management routines, conversion routines, and string manipulation routines. The lowest layer, operating system interface routines, is documented for adaptation to the target environment.

FORTRAN COMPILER FOR THE i486™ MICROPROCESSOR

The Fortran-386 translator meets the ANSI Fortran 77 Language Subset Specification. This compatibility assures portability of existing Fortran programs and shortens the development process when moving Fortran databases off of mini or mainframe computers and onto i486 microprocessor-based systems.

Fortran-386 provides extensive support for numeric processing tasks and applications, with features such as:

- Support for floating-point data types including single, double, and double extended precision; complex and double complex data types; as well as integer, signed, and logical data types
- Support for the proposed REALMATH IEEE floating point standard
- Support for Fortran language extensions endorsed by the U.S. Department of Defense
- Support of Intel's standard object code format for symbolic debug, and for linking object modules from other Intel 386/486 languages

SOFTWARE UTILITIES FOR THE i486™ MICROPROCESSOR

The RLL-386/486 software utilities are a cohesive set of software design aids for programming the i486™ microprocessor system. The package enables system programmers to design protected, multi-user and multi-tasking operating system software. RLL-386/486 provides for a variety of functions— from linking separate modules, building an object library, linking in floating point tasks, learning the i486 architecture with a series of templates, or building a task to execute under protected mode.

The RLL-386/486 software utilities package includes a program binder for linking separately compiled modules together, a system builder for configuring protected multiple-task systems, a floating point and numeric support library, a cross-reference mapper listing symbolic information, a program librarian, and a conversion utility to download hex format for PROM programming.

ICD-486 IN-CIRCUIT DEBUGGER

ICD-486 In-Circuit Debugger provides sophisticated real-time hardware and software debug capabilities for i486 microprocessor based designs. The user can run at the full speed of the i486 CPU, ensuring that the elusive timing bugs will be found. And because ICD-486 is completely non-intrusive, no modifications will be required in the final target system.

ICD-486 is the first development tool which allows users to debug high speed cached applications at the full speed of the processor. ICD-486 embodies exclusive technology, providing symbolic access to internal processor states that would not be accessible in any other way. With Intel's exclusive technology, users can be assured that the ICD-486 provides complete accuracy when debugging cached applications in real time.

ICD-486 can be used by both hardware and real-time software developers, at any stage of development. Early in the development process, ICD-486 allows prototype development and software debugging. Later in the design cycle, the ICD-486 can be used when integrating hardware and software modules. With symbolic debugging, memory locations can be examined or modified using symbolic references to the original program, such as a procedure or variable name, line number, or program label.

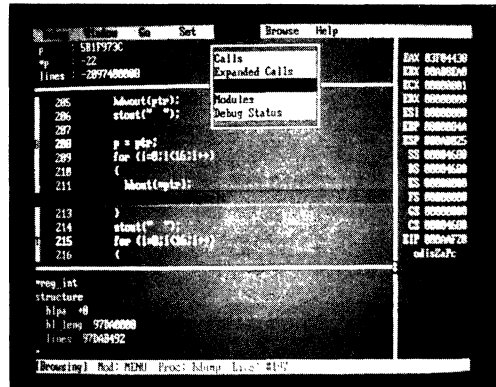


Figure 2. DB386 Screen Display

DOS-RESIDENT SOFTWARE DEBUGGER

DB386 is an on-host software execution environment with source-level symbolic debug capabilities for object modules produced by Intel's ASM-386/486, C-386/486, PL/M-386/486, and Fortran-386 translators. This software debug environment allows i486 software applications to be executed and debugged directly on a 386 or 386SX™ microprocessor based PC, without any additional target hardware required. With Intel's standard windowed human interface, users can focus their efforts on finding bugs rather than spending time learning and manipulating the debug environment.

- **Supports key features of the i486 architecture.**

A run-time interface allows protected-mode programs to be executed directly on a 386- or 386SX-based PC.

- **Ease of learning.** Drop-down menus make the tool easy to learn for new or casual users. A command line interface is also provided for more complex problems.
- **Extensive debug modes.** Watch windows (display user-specified variables), trace points, and breakpoints (including fixed, temporary, and conditional) can be set and modified as needed, even during a debug session.
- **See into the i486 microprocessor.** The user can browse source and call stacks, observe processor registers, and access watch window variables by either the pull down menu or by a single keystroke using the function keys. An easy-to-use disassembler and single-line assembler can also speed the debug process.
- **Full debug symbolics for maximum productivity.** The user need not know whether a variable is an unsigned integer, a real, or a structure. The debugger utilizes the wealth of typing information available in Intel languages to display program variables in their respective type formats.

SERVICE, SUPPORT, AND TRAINING

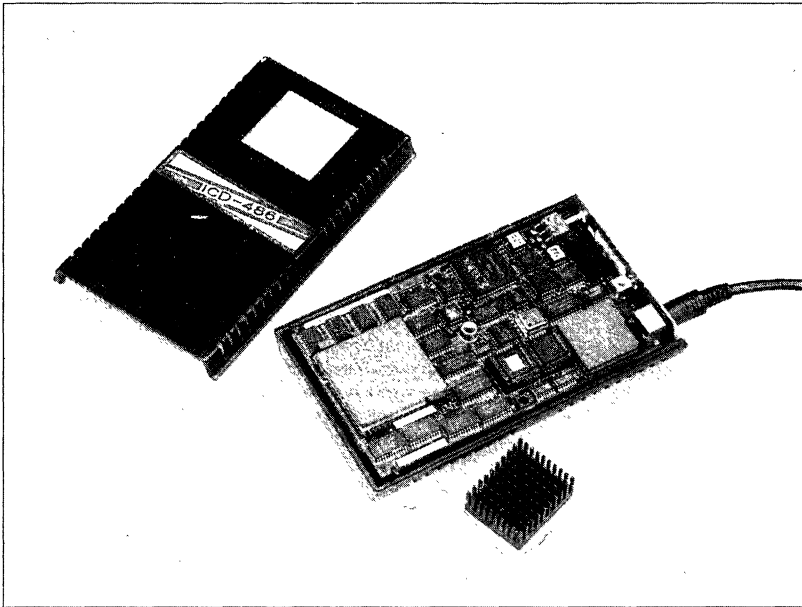
To augment its development tools, Intel offers a full array of seminars, classes, and workshops, field application engineering expertise, hotline technical support, and on-site service.

i486, Intel386, 386, 386SX, 376, ICE, and iSBC are trademarks of Intel Corporation. VAX and VMS are registered trademarks of Digital Equipment Corporation.

ORDERING INFORMATION

For direct information on Intel's Development Tools, or for the number of your nearest sales office or distributor, call 800-874-6835 (U.S.). For information or literature on additional Intel products, call 800-548-4725 (U.S. and Canada).

ICD-486 IN-CIRCUIT DEBUGGER



i486™ MICROPROCESSOR IN-CIRCUIT DEBUGGER

Intel's ICD-486, the in-circuit debugger for the i486™ microprocessor, represents a new generation of in-circuit emulation technology. From the inventor of the microprocessor comes a development tool that delivers complete access to the i486 architecture.

ICD-486 is the first development tool which allows users to debug high speed cached applications at the full speed of the processor. ICD-486 embodies exclusive technology, providing symbolic access to internal processor states that would not be accessible in any other way. With Intel's exclusive technology, users can be assured that the ICD-486 provides complete accuracy when debugging cached applications in real time.

FEATURES

- Real time emulation at the full speed of the i486 microprocessor
- Full development and debug support for the i486 microprocessor on-chip caching and numerics
- Programming support for the i486 microprocessor real mode **and** native protected mode
- Non-intrusive operation, allowing the target system to be debugged without modification
- Ability to set up to 4 hardware and 16 software breakpoints on execution addresses, data read, or data access
- A sync-out line for hooking the ICD-486 up to a high speed logic analyzer to provide trace information
- Provides full symbolic information to display and modify all registers

FULL-SPEED DEBUG AND DEVELOPMENT

The ICD-486 In-Circuit Debugger provides sophisticated real-time hardware and software debug capabilities for i486 microprocessor based designs. The user can run at the full speed of the i486 CPU, ensuring that the elusive timing bugs will be found. And because ICD-486 is non-intrusive, the target system being debugged can be the same as the final target system.

DEBUG CACHED APPLICATIONS

Until now, it has been extremely difficult to accurately debug high speed cached microprocessor applications. However, by incorporating Intel's exclusive technology, the ICD-486 allows users to debug applications which utilize the on-chip caching capabilities of the i486 microprocessor. This is not just a statistical reconstruction of the cache—the ICD-486 provides complete accuracy when debugging applications regardless of whether cache is on or off.

IDEAL FOR ALL STAGES OF DEVELOPMENT

ICD-486 can be used by both hardware and real-time software developers, at any stage of development. Early in the development process, ICD-486 allows prototype development and software debugging. Later in the design cycle, the ICD-486 can be used when integrating hardware and software modules.

SPEEDING DEVELOPMENT WITH SYMBOLICS

With symbolic debugging, memory locations can be examined or modified using symbolic references to the original program, such as a procedure or variable name, line number, or program label. Microprocessor data structures, such as registers, descriptor tables, and page tables, can also be examined and modified using symbolic names rather than via cumbersome linear or physical addresses. Optimal symbolic debugging can be achieved when using the ICD-486 with Intel languages, which produce not only address and memory locations but also variable and procedure typing information.

THE COMPLETE STORY

For advanced hardware debugging, the ICD-486 has been designed to work with high speed logic analyzers. The ICD-486 ships with a Logic Analyzer Interface board which provides the control signals to trigger the logic analyzer. With a user-supplied software interface, the ICD-486 and logic analyzer can work in combination to monitor or recognize bus activity and to gather execution trace.

SOFTWARE COMPLETES THE SYSTEM

Intel provides a comprehensive software development environment to complement the ICD-486, delivering the most complete 32-bit microprocessor development environment available from a single vendor.

Intel's i486 software development tools offer a broad choice of languages with object code compatibility so performance can be maximized by using different languages for specific tasks. Architectural extensions in the high level languages allow hardware features such as interrupts, I/O, or flags to be controlled directly, avoiding the tedium and overhead of assembly routines.

Intel's software environment includes a sophisticated software debugger and execution environment, allowing i486 software applications to be tested and debugged directly on a standard 386™ microprocessor-based PC. To provide full access to the power of the i486 architecture, the software portfolio incorporates a unique, sophisticated, and very powerful system builder, simplifying the generation of protected mode systems. To further reduce the task of integrating software into the final target configuration, Intel i486 microprocessor software tools produce code which can be directly downloaded in target system ROM or can be converted into standard hex code.

THE RIGHT TOOL FOR THE JOB

ICD-486, the new generation of in-circuit emulation technology, is the right tool to use when your product development schedules are tight and your product quality requirements are high. Intel's exclusive technology allows you to debug cached applications at the full speed of the i486 processor, and the symbolic debug information can vastly improve your productivity.

WORLDWIDE, WORLD CLASS SERVICES

Augmenting Intel's i486 microprocessor development tools is a full array of seminars, classes, and workshops; on-site consulting services; field application engineering expertise; telephone hotline support; and software and hardware maintenance contracts.

ORDERING INFORMATION

- | | |
|----------------|---|
| ICD48625D | In-circuit debugger for the i486 microprocessor. Operates to 25 MHz. Includes hardware debug module, power supply, isolation board, stand-alone self-test board, logic analyzer interface board, user documentation, and DOS host software and interface cable. |
| ICD48625CON33D | Identical to the ICD48625D, but also includes a guaranteed upgrade to 33 MHz i486 microprocessor support when available. |

For direct information on Intel's Development Tools, or for the number of your nearest sales office or distributor, call 800-874-6835 (U.S.). For information or literature on additional Intel products, call 800-548-4725 (U.S. and Canada).



IN-CIRCUIT EMULATOR FOR THE 8086/80186/80286 FAMILY OF MICROPROCESSORS

The I²ICE™ In-Circuit Emulator is a high-performance, cost-effective debug environment for developing systems with the Intel 8086/80186/80286 family of microprocessors. With 10 MHz emulation, a window-oriented user interface, and compatibility with Intel's iPAT™ Performance Analysis Tool, the I²ICE Emulator gives you unmatched speed and control over all phases of hardware/software debug.

FEATURES

- Emulation speeds up to 10 MHz with 8086/88, 80186/188 and 80286 microprocessors
- 8087 and 80287 numeric coprocessor support
- Hosted on IBM PC AT*, AT BIOS, or compatibles
- ICEVIEW™ window-oriented user interface with pull-down menus and context-sensitive help
- Source and symbol display using all Intel languages
- 1K frame bus and execution trace buffer
- Symbolic debugging for flexible access to memory location and program variables
- Flexible breakpointing for quick problem isolation
- Memory expandable to 288K with zero wait states
- Worldwide service and support
- iPAT option for software speed tuning

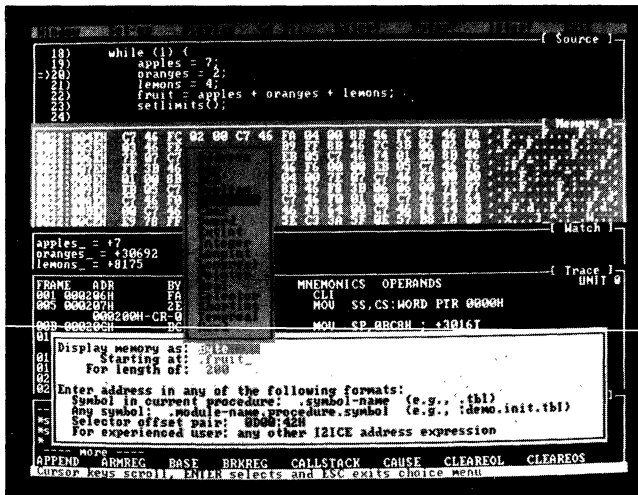


Plate 1. An example of the ICEVIEW™ user interface showing source, memory, watch, and trace.

ONE TOOL FOR THE ENTIRE DEVELOPMENT PROCESS

The I2ICE Emulator allows hardware and software design to proceed simultaneously, so you can develop software even before prototype hardware is available. With 32K of zero wait-state mappable memory (and an additional 256K with optional memory boards), you can use the I2ICE Emulator to debug at any stage of the development cycle: hardware development, software development, system integration or system test.

HIGH-SPEED, REAL-TIME EMULATION

The I2ICE Emulator delivers full-speed, real-time emulation at speeds up to 10 MHz. Based on Intel's exclusive microprocessor technology, the I2ICE Emulator matches each chip's electrical and timing characteristics without memory or interrupt intrusions, ensuring design accuracy and eliminating surprises. The performance of your prototype is the performance you can expect from your final product.

EASY-TO-USE ICEVIEW™ INTERFACE

The ICEVIEW interface makes the I2ICE Emulator easy to learn and use by providing easy access to application information and ICE functions. Pull-down menus and windows boost productivity for both new and experienced users. Multiple on-screen windows allow you to access the source display, execution trace, register, and other important information, all at the same time. You can watch the information change as you modify and step through your program. You can even customize window size and screen positions.

A command line interface is also available with syntax checking and context-sensitive prompts. ICEVIEW works with monochrome, CGA and the latest EGA color displays.

SYMBOLIC DEBUG SPEEDS DEVELOPMENT

The extensive debug symbolics generated by the Intel 8086 and 80286 assemblers and compilers can increase your development productivity. Symbolics with automatic formatting are available for all primitive types, regardless of whether the variables are globals, locals (stack-resident) or pointers. The virtual symbol table supports all symbolics, even in very large programs. Aliasing can be used to reduce keystrokes and save time.

POWERFUL BREAK AND TRACE CAPABILITY FOR FAST PROBLEM ISOLATION

The I2ICE Emulator allows up to eight simultaneous break/trace conditions to be set (four execution, four bus), a timesaver when solving hardware/software integration problems. Break and trace points can be set on specified line numbers, on procedures, or on symbolic data events, such as writing a variable to a value or range of values. You can break or trace on specific hardware events, such as a read or write to a specific address, data or I/O port, or on a combination of events.

MULTIPROCESSOR, PROTECTED MODE, AND COPROCESSOR SUPPORT

Up to four I2ICE systems can be linked and controlled simultaneously from one PC host, enabling you to debug multiprocessor systems. The I2ICE Emulator with an 80286 probe supports all 80286 protected mode capabilities. It also supports the 8087 and 80287 numeric coprocessors.

iPAT™ FOR SOFTWARE PERFORMANCE AND CODE COVERAGE ANALYSIS

The I²ICE Emulator interfaces to Intel's iPAT Performance Analysis Tool for examining software execution speeds and code coverage in real time. iPAT displays critical performance data about your code in easy-to-understand histograms and tables. Elusive bottlenecks are readily seen, allowing you to focus your attention to get the most performance out of your product.

iPAT also performs code execution coverage, letting you perform product evaluations faster and more effectively. iPAT pinpoints areas in your code either executed or not executed according to specific conditions, taking the guesswork out of software evaluations.

EASY INTERFACE TO EXTERNAL INSTRUMENTS

The I²ICE system includes external emulation clips and software support for setting breakpoints, tracepoints and arm/disarm conditions on external events, making it easy to connect external logic analyzers and signal generators. You can debug complex hardware/software interactions with a high level of productivity.

WORLDWIDE SERVICE AND SUPPORT

The I²ICE Emulator is supported by Intel's worldwide service and support organization. In addition to an extended warranty, you can choose from hotline support, on-site systems engineering assistance, and a variety of hands-on training workshops.

SPECIFICATIONS

HOST REQUIREMENTS

IBM PC/AT or 100% PC AT BIOS compatible
 DOS 3.1 or later
 640K bytes of memory
 360K bytes or 1.2 MB floppy disk drive
 Hard disk drive
 Monochrome, CGA or EGA monitor (EGA recommended)

PHYSICAL DESCRIPTION

Unit	Width		Height		Length	
	cm	in	cm	in	cm	in
I ² ICE chassis	43.2	17.0	21.0	8.25	61.3	24.13
Probe base	21.6	8.5	7.6	3.0	25.4	10.0

Host/chassis cable 15 ft. (4.6 m)

ELECTRICAL CHARACTERISTICS

90-132 V or 180-264 V (selectable)
 47-63 Hz
 12 amps (AC)

ENVIRONMENTAL SPECIFICATIONS

Operating temperature: 0-40°C (32-104°F) ambient
 Operating humidity: Maximum of 85% relative humidity, non-condensing

ORDERING INFORMATION

Kit Code	Contents
plll010KITD	I ² ICE system 10 MHz 8086/8088 support kit for IBM PC host. Includes probe, chassis, and host interface module and software.
plll111KITD	I ² ICE system 10 MHz 80186 support kit for IBM PC host. Includes probe, chassis, host interface module and software. Note: For 80188 support, the lll198 option below must also be ordered.
lll198	10 MHz 80188 support conversion kit to convert 80186 probe to 80188 probe.
plll212KITD	I ² ICE system, 10 MHz 80286 support kit for IBM PC AT host. Includes probe, chassis, host interface module and software.
lll010PATC86D	I ² ICE system 10 MHz 8086/8088 support kit with iPAT Performance Analysis Tool for PC AT host. Includes I ² ICE probe, chassis, host interface module, iPAT tool option, cables and software. Also includes iC-86 compiler, 86 Macro Assembler, utilities, and AEDIT text editor.
lll111PATC86D	As above for 10 MHz 80186 support.
lll212PATC86D	As above for 10 MHz 80286 support. Note: C-286 and RLL-286 and ASM-286 must be ordered separately.
954D	I ² ICE PC AT host software. Includes ICEVIEW™ windowed human interface.

Note: I²ICE probes, chassis, software, cables and iPAT options are available separately.

ICE™ -186 IN-CIRCUIT EMULATOR



HIGH PERFORMANCE REAL-TIME EMULATION

Intel's ICE-186 emulator delivers real-time emulation for the 80C186 microprocessor at speeds up to 12.5 MHz. The in-circuit emulator is a versatile and efficient tool for developing, debugging and testing products designed with the Intel 80C186 microprocessor. The ICE-186 emulator provides real time, full speed emulation in a user's system. Popular features such as symbolic debug, 2K bytes trace memory, and single-step program execution are standard on the ICE-186 emulator. Intel provides a complete development environment using assembler (ASM86) as well as high-level languages such as Intel's iC86, PL/M86, Pascal 86 and Fortran 86 to accelerate development schedules.

The ICE-186 emulator supports a subset of the 80C186 features at 12.5 MHz and at the TTL level characteristics of the component. The emulator is hosted on IBM's Personal Computer AT, already available as a standard development solution in most of today's engineering environments. The ICE-186 emulator operates in prototype or standalone mode, allowing software development and debug before a prototype system is available. The ICE-186 emulator is ideally suited for developing real-time applications such as industrial automation, computer peripherals, communications, office automation, or other applications requiring the full power of the 12.5 MHz 80C186 microprocessor.

ICE™-186 FEATURES

- Full 12.5 MHz Emulation Speed
- 2K Frames Deep Trace Memory
- Two-Level Breakpoints with Occurrence Counters
- Single-Step Capability
- 128K Bytes Zero Wait-State Mapped Memory
- Supports DRAM Refresh
- High-Level Language Support
- Symbolic Debug
- RS-232-C and GPIB Communication Links
- Crystal Power Accessory
- Interface for Intel Performance Analysis Tool (iPAT)
- Interface for Optional General Purpose Logic Analyzer
- Tutorial Software
- Complete Intel Service and Support

intel

HIGHEST EMULATION SPEED AVAILABLE TODAY

The ICE-186 emulator supports development and debug of time-critical hardware and software using Intel's 12.5 MHz 80C186 microprocessor.

RETRACE SOFTWARE TRACKS

This emulator captures up to 2,048 frames of processor activity, including both execution and data bus activity. With this trace memory, large blocks of program code can be traced in real time and viewed for program flow and behavior characteristics.

HARDWARE BREAKPOINTS FOR COMPLEX DEBUG

User-defined "TIL-THEN" breakpoint statements stop emulation at specific execution addresses or bus events. During the hardware and software integration phase, breakpoint statements can be defined as execution addresses and/or bus addresses and/or bus access types such as memory and I/O reads or writes. Additionally, event counters provide another level of breakpoint control for sophisticated state machine constructs used to specify emulation breakpoints/tracepoints.

SMALL OR LARGE STEPS

A stepping command can be used to view program execution one instruction at a time or in preset instruction blocks. When used in conjunction with symbolic debug, code execution can be monitored quickly and precisely.

DEBUG CODE WITHOUT A PROTOTYPE

Even before prototype hardware is available, the ICE-186 emulator working in conjunction with the Crystal Power Accessory (CPA) creates a "virtual" application environment. 128K bytes of zero wait-state memory is available for mapped memory and I/O resource addressing in 4K increments. The CPA provides emulator diagnostics as well as the ability to use the emulator without a prototype.

DON'T LOSE MEMORY

The ICE-186 emulator continues DRAM refresh signals even when emulation has been halted, thus ensuring DRAM memory will not be lost. During interrogation mode the ICE-186 emulator will keep the timers functioning and correctly respond to interrupts in real-time.

HIGH LEVEL LANGUAGE SUPPORT OPTIMIZED FOR INTEL TOOLS

The ICE-186 supports emulation for programs written in Intel's ASM86 or any of Intel's high-level languages:

PL/M-86	Fortran-86
Pascal-86	C-86

These languages are optimized for the Intel 80186/80188 component architectures to deliver a tightly integrated, high performance development environment.

USER-FRIENDLY SYMBOLICS AID IN DEBUG

Symbolics allow access to program symbols by name rather than cumbersome physical addresses. Symbolic debug speeds the debugging process by reducing reliance on memory maps. In a dynamic development process, user variables can be used as parameters for ICE-186 commands resulting in a consistent debug environment.

SUPPORTS FAST BREAKS

"Fastbreaks" is a feature which allows the emulation processor to halt, access memory, and return to emulation as quickly as possible. A fastbreak never takes more than 5625 clock cycles (most types of fastbreaks are considerably less). This feature is particularly useful in embedded applications.

MULTIPLE HIGH-SPEED COMMUNICATION LINKS

Two communication links are available for use in conjunction with the host IBM PC AT. The ICE-186 emulator uses either serial (RS-232-C) or a parallel (GPIB) link. A user supplied National Instruments (IEEE-488) GPIB communication board provides parallel transfers at rates up to 300K bytes per second.

SOFTWARE ANALYSIS (iPAT)

Intel's Performance Analysis Tool (iPAT) is designed to increase team productivity with features like interrupt latency measurement, code coverage analysis and software module performance analysis. These features enable the user to design reliable, high performance embedded control products. The ICE-186 emulator has an external 60 pin connector for iPAT.

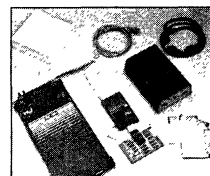
BUILT-IN SUPPORT FOR LOGIC ANALYSIS

General-purpose logic analyzers can be used in conjunction with the ICE-186 to provide detailed timing of specific events. The ICE-186 emulator provides an external sync signal for triggering logic analysis, making complex trigger sequence programming easy. An additional 60 pin connector is included for the logic analyzer.

WORLDWIDE SERVICE AND SUPPORT

The ICE-186 emulator is supported by Intel's worldwide service and support organization. Total hardware and software support is available including a hotline number when the need is there.

Note: This emulator does not support use of the 8087.



SPECIFICATIONS

PERSONAL COMPUTER REQUIREMENTS

The ICE-186 emulator is hosted on an IBM PC AT. The emulator has been tested and evaluated on an IBM PC AT. The PC AT must meet the following minimum requirements:

- 640K Bytes of Memory
- Intel Above Board with at Least 1M Byte of Expansion Memory
- One 360K Bytes or One 1.2M Bytes floppy Disk Drive
- One 20M Bytes Fixed-Disk Drive
- PC DOS 3.2 or Later
- A serial Port (COM1 or COM2) Supporting Minimally at 9600 Baud Data Transfers, or a National Instruments GPIB-PC2A board.
- IBM PC AT BIOS

PHYSICAL DESCRIPTION AND CHARACTERISTICS

The ICE-186 Emulator consists of the following components:

Unit	Width		Height		Length	
	Inches	Cm.	Inches	Cm.	Inches	Cm.
Emulator						
Control Unit	10.40	26.40	1.70	4.30	20.70	52.60
Power Supply	7.60	19.00	4.15	10.70	11.00	27.90
User Probe	3.70	9.40	.65	1.60	7.00	17.80
User Cable/ Pllc					22.00	55.90
Hinge Cable					3.40	8.60
Crystal Power						
Accessory CPA Power	4.30	10.90	.60	1.50	6.70	17.00
Cable					9.00	22.90

ELECTRICAL CONSIDERATIONS

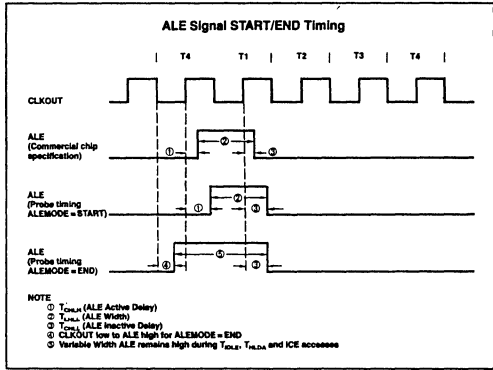
I_{CC} 1050mA
 I_{IH} 70μA Max.
 I_{IL} -1.5mA Max
 I_{OH} -1.0mA Max.

TIMING CONSIDERATIONS

<i>ICE-186 User AC Differences</i>						
<i>Symbol</i>	<i>Parameter</i>	<i>COMPONENT SPEC</i>		<i>ICE-186 SPEC</i>		
		<i>Min.</i>	<i>Max.</i>	<i>Min.</i>	<i>Max.</i>	
TD _{VCL}	Data in Setup (A/D)	15		24		
TA _{RYCH}	Async Ready (ARDY) Resolution Transition Setup Time	15		23		
TS _{RYCL}	Synchronous Ready (SRDY) Transition Setup Time	15		25		
TH _{VCL}	HOLD Setup	15		32		
TI _{NVCH}	NMI	15		32		
	/TEST	15		31		
	INTR,TIMERIN Setup Time	15		17		
TI _{NVCL}	DRQ0, DRQ1, Setup Time	15		19		
TC _{LAZ}	Address Float Delay					
	READ Cycles	TC _{LAX}	25	5	36	
	INTA cycles	TC _{LAX}	25	0	25	
	HLDA	TC _{LAX}	25	10	50	
TL _{HLL}	ALE Width (min)	TC _{LCL}	30	TC _{LCL}	32	
TC _{HLLH}	ALE Active Delay*		25		42	
(N/A)	CLKOUT Low to ALE Active**		(N/A)		19	
TC _{HLL}	ALE Inactive Delay		25		40	
TL _{LAX}	Address Hold to ALE Inactive (min)	TC _{HCL}	15	TC _{HCL}	28	
TC _{WCTY}	Control Inactive Delay	5	37	1	40	
TA _{ZRL}	Address Float to /RD Active	0		-30		
TA _{VLL}	Address VAlid to ALE Low (min)	TC _{LCH}	15	TC _{LCH}	19	
TC _{HQSV}	Que Status Delay		28		35	
TD _{XDL}	/DEN Inactive to DT /R Low	0		-7		
TC _{ICO}	CLKIN to CLKOUT Skew		21		37	

Consult User Guide for Additional Specifications.

*Applies only when the ALEMODE variable is set to START.
 **Applies only when the ALEMODE variable is set to END.



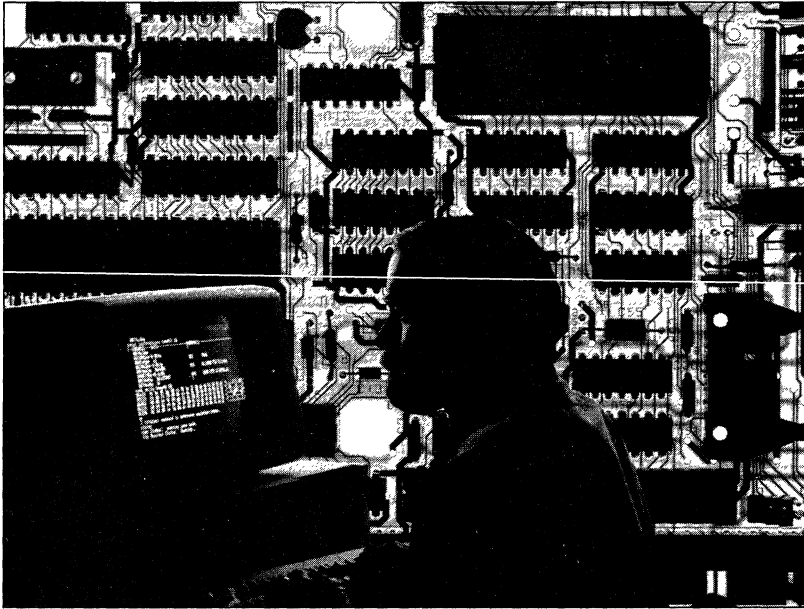
ENVIRONMENTAL SPECIFICATIONS

Operating Temperature 10°C to 40°C Ambient
Storage Temperature -40°C to 70°C

ORDERING INFORMATION

- ICE186 ICE-186 System including ICE software (Requires DOS 3.XX PC AT with Above Board)
- ICE 186AB ICE 186 with Above Board included
- ICE186IPAT ICE-186 System including ICE S/W packages and the IPAT system (Requires DOS 3.XX PC AT with Above Board)
- D86ASM86NL 86 macro assembler 86 builder/binder/mapper utilities for DOS 3.XX.
- D86C86NL 86 C compiler and run time libraries for DOS 3.XX.
- D86PAS86NL 86 Pascal Compiler for DOS 3.XX.
- D86PLM86NL 86 PL/M compiler for DOS 3.XX.
- D86FOR86NL 86 Fortran compiler for DOS 3.XX.
- ICEPAT KIT iPAT Kit (Performance Analysis Tool) for ICE 186
- ICEXONCE Adapter for on-circuit emulation
- ICEXLCC Adapter for LCC component
- ICEXPGA Adapter for PGA component

ICE™ -188 IN-CIRCUIT EMULATOR



HIGH PERFORMANCE REAL-TIME EMULATION

Intel's ICE-188 emulator delivers real-time emulation for the 80C188 microprocessor at speeds up to 12.5 MHz. The in-circuit emulator is a versatile and efficient tool for developing, debugging and testing products designed with the Intel 80C188 microprocessor. The ICE-188 emulator provides real time, full speed emulation in a user's system. Popular features such as symbolic debug, 2K bytes trace memory, and single-step program execution are standard on the ICE-188 emulator. Intel provides a complete development environment using assembler (ASM86) as well as high-level languages such as Intel's IC86, PL/M86, Pascal 86 and Fortran 86 to accelerate development schedules.

The ICE-188 emulator supports a subset of the 80C188 features at 12.5 MHz and at the TTL level characteristics of the component. The emulator is hosted on IBM's Personal Computer AT, already available as a standard development solution in most of today's engineering environments. The ICE-188 emulator operates in prototype or standalone mode, allowing software development and debug before a prototype system is available. The ICE-188 emulator is ideally suited for developing real-time applications such as industrial automation, computer peripherals, communications, office automation, or other applications requiring the full power of the 12.5 MHz 80C188 microprocessor.

ICE™ -188 FEATURES

- Full 12.5 MHz Emulation Speed
- 2K Frames Deep Trace Memory
- Two-Level Breakpoints with Occurrence Counters
- Single-Step Capability
- 128K Bytes Zero Wait-State Mapped Memory
- Supports DRAM Refresh
- High-Level Language Support
- Symbolic Debug
- RS-232-C and GPIB Communication Links
- Crystal Power Accessory
- Interface for Intel Performance Analysis Tool (IPAT)
- Interface for Optional General Purpose Logic Analyzer
- Tutorial Software
- Complete Intel Service and Support



HIGHEST EMULATION SPEED AVAILABLE TODAY

The ICE-188 emulator supports development and debug of time-critical hardware and software using Intel's 12.5 MHz 80C188 microprocessor.

RETRACE SOFTWARE TRACKS

This emulator captures up to 2,048 frames of processor activity, including both execution and data bus activity. With this trace memory, large blocks of program code can be traced in real time and viewed for program flow and behavior characteristics.

HARDWARE BREAKPOINTS FOR COMPLEX DEBUG

User-defined "TIL-THEN" breakpoint statements stop emulation at specific execution addresses or bus events. During the hardware and software integration phase, breakpoint statements can be defined as execution addresses and/or bus addresses and/or bus access types such as memory and I/O reads or writes. Additionally, event counters provide another level of breakpoint control for sophisticated state machine constructs used to specify emulation breakpoints/tracepoints.

SMALL OR LARGE STEPS

A stepping command can be used to view program execution one instruction at a time or in preset instruction blocks. When used in conjunction with symbolic debug, code execution can be monitored quickly and precisely.

DEBUG CODE WITHOUT A PROTOTYPE

Even before prototype hardware is available, the ICE-188 emulator working in conjunction with the Crystal Power Accessory (CPA) creates a "virtual" application environment. 128K bytes of zero wait-state memory is available for mapped memory and I/O resource addressing in 4K increments. The CPA provides emulator diagnostics as well as the ability to use the emulator without a prototype.

DON'T LOSE MEMORY

The ICE-188 emulator continues DRAM refresh signals even when emulation has been halted, thus ensuring DRAM memory will not be lost. During interrogation mode the ICE-188 emulator will keep the timers functioning and correctly respond to interrupts in real-time.

HIGH LEVEL LANGUAGE SUPPORT OPTIMIZED FOR INTEL TOOLS

The ICE-188 supports emulation for programs written in Intel's ASM86 or any of Intel's high-level languages:

PL/M-86	Fortran-86
Pascal-86	C-86

These languages are optimized for the Intel 80186/80188 component architectures to deliver a tightly integrated, high performance development environment.

USER-FRIENDLY SYMBOLICS AID IN DEBUG

Symbolics allow access to program symbols by name rather than cumbersome physical addresses. Symbolic debug speeds the debugging process by reducing reliance on memory maps. In a dynamic development process, user variables can be used as parameters for ICE-188 commands resulting in a consistent debug environment.

SUPPORTS FAST BREAKS

"Fastbreaks" is a feature which allows the emulation processor to halt, access memory, and return to emulation as quickly as possible. A fastbreak never takes more than 5625 clock cycles (most types of fastbreaks are considerably less). This feature is particularly useful in embedded applications.

MULTIPLE HIGH-SPEED COMMUNICATION LINKS

Two communication links are available for use in conjunction with the host IBM PC AT. The ICE-188 emulator uses either serial (RS-232-C) or a parallel (GPIB) link. A user supplied National Instruments (IEEE-488) GPIB communication board provides parallel transfers at rates up to 300K bytes per second.

SOFTWARE ANALYSIS (IPAT)

Intel's Performance Analysis Tool (iPAT) is designed to increase team productivity with features like interrupt latency measurement, code coverage analysis and software module performance analysis. These features enable the user to design reliable, high performance embedded control products. The ICE-188 emulator has an external 60 pin connector for iPAT.

BUILT-IN SUPPORT FOR LOGIC ANALYSIS

General-purpose logic analyzers can be used in conjunction with the ICE-188 to provide detailed timing of specific events. The ICE-188 emulator provides an external sync signal for triggering logic analysis, making complex trigger sequence programming easy. An additional 60 pin connector is included for the logic analyzer.

WORLDWIDE SERVICE AND SUPPORT

The ICE-188 emulator is supported by Intel's worldwide service and support organization. Total hardware and software support is available including a hotline number when the need is there.

Note: This emulator does not support use of the 8087.

SPECIFICATIONS

PERSONAL COMPUTER REQUIREMENTS

The ICE-188 emulator is hosted on an IBM PC AT. The emulator has been tested and evaluated on an IBM PC AT. The PC AT must meet the following minimum requirements:

- 640K Bytes of Memory
- Intel Above Board with at Least 1M Byte of Expansion Memory
- One 360K Bytes or One 1.2M Bytes floppy Disk Drive
- One 20M Bytes Fixed-Disk Drive
- PC DOS 3.2 or Later
- A serial Port (COM1 or COM2) Supporting Minimally at 9600 Baud Data Transfers, or a National Instruments GPIB-PC2A board.
- IBM PC AT BIOS

PHYSICAL DESCRIPTION AND CHARACTERISTICS

The ICE-188 Emulator consists of the following components:

Unit	Width		Height		Length	
	Inches	Cm.	Inches	Cm.	Inches	Cm.
Emulator						
Control Unit	10.40	26.40	1.70	4.30	20.70	52.60
Power Supply	7.60	19.00	4.15	10.70	11.00	27.90
User Probe	3.70	9.40	.65	1.60	7.00	17.80
User Cable/ Plcc					22.00	55.90
Hinge Cable					3.40	8.60
Crystal Power						
Accessory CPA Power Cable	4.30	10.90	.60	1.50	6.70	17.00
					9.00	22.90

ELECTRICAL CONSIDERATIONS

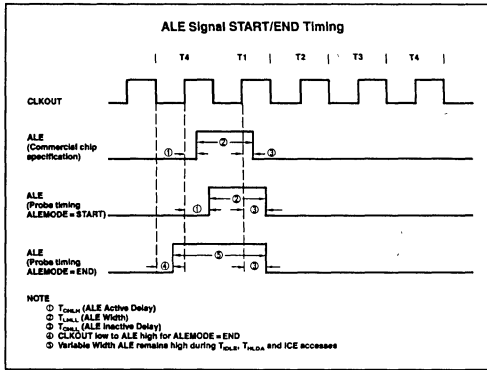
I_{CC} 1050mA
 I_{IH} 70μA Max.
 I_{IL} -1.5mA Max
 I_{OH} -1.0mA Max.

TIMING CONSIDERATIONS

ICE-188 User AC Differences						
Symbol	Parameter	COMPONENT SPEC		ICE-188 SPEC		
		Min.	Max.	Min.	Max.	
TD _{VCL}	Data in Setup (A/D)	15		24		
TA _{RYCH}	Async Ready (ARDY) Resolution Transition Setup Time	15		23		
TS _{RYCL}	Synchronous Ready (SRDY) Transition Setup Time	15		25		
TH _{VCL}	HOLD Setup	15		32		
TI _{NVCH}	NMI	15		32		
	/TEST	15		31		
	INTR, TIMERIN Setup Time	15		17		
TI _{NVCL}	DRQ0, DRQ1, Setup Time	15		19		
TC _{LAX}	Address Float Delay					
	READ Cycles	TC _{LAX}	25	5	36	
	INTA cycles	TC _{LAX}	25	0	25	
	HLDA	TC _{LAX}	25	10	50	
TL _{HLL}	ALE Width (min)	TC _{LCL}	30	TC _{LCL}	32	
TC _{HLH}	ALE Active Delay*		25		42	
	(N/A) CLKOUT Low to ALE Active**	(N/A)			19	
TC _{HLL}	ALE Inactive Delay		25		40	
TL _{LAX}	Address Hold to ALE Inactive (min)	TC _{HCL}	15	TC _{HCL}	28	
TC _{VCIX}	Control Inactive Delay	5	37	1	40	
TA _{ZRL}	Address Float to /RD Active	0		-30		
TA _{VLL}	Address VAlid to ALE Low (min)	TC _{LCH}	15	TC _{LCH}	19	
TC _{HOSV}	Que Status Delay		28		35	
TD _{XDL}	/DEN Inactive to DT /R Low	0		-7		
TC _{ICD}	CLKIN to CLKOUT Skew		21		37	

Consult User Guide for Additional Specifications.

*Applies only when the ALEMODE variable is set to START.
 **Applies only when the ALEMODE variable is set to END.



ENVIRONMENTAL SPECIFICATIONS

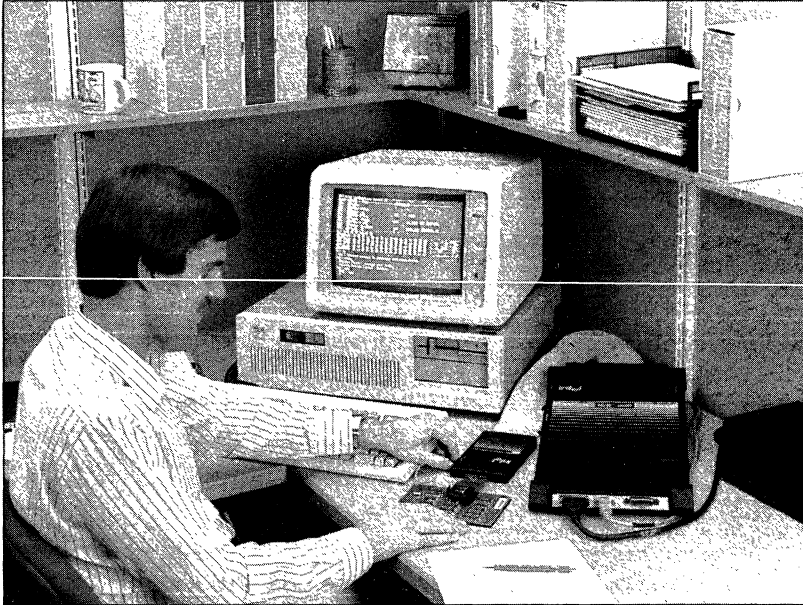
Operating Temperature 10°C to 40°C Ambient
Storage Temperature -40°C to 70°C

ORDERING INFORMATION

ICE188	ICE-188 System including ICE software (Requires DOS 3.XX PC AT with Above Board)
ICE 188AB	ICE 188 with Above Board included
D86ASM86NL	86 macro assembler 86 builder/binder/mapper utilities for DOS 3.XX.
D86C86NL	86 C compiler and run time libraries for DOS 3.XX.
D86PAS86NL	86 Pascal Compiler for DOS 3.XX.
D86PLM86NL	86 PL/M compiler for DOS 3.XX.
D86FOR86NL	86 Fortran compiler for DOS 3.XX.
ICEPAT KIT	iPAT Kit (Performance Analysis Tool) for ICE 188
ICEXONCE	Adapter for on-circuit emulation
ICEXLCC	Adapter for LCC component
ICEXPGA	Adapter for PGA component
UP188	User probe to convert ICE-186 to support 80C188 component

For direct information on Intel's Development Tools, or for the number of your nearest sales office or distributor, call 800-874-6835 (U.S.). For information or literature on additional Intel products, call 800-548-4725 (U.S. and Canada).

ICE-286 IN-CIRCUIT EMULATOR



HIGH PERFORMANCE REAL-TIME EMULATION

Intel's ICE-286 emulator delivers real-time emulation for the 80286 microprocessor at speeds up to 12.5 MHz. The in-circuit emulator is a versatile and efficient tool for developing, debugging and testing products designed with the Intel 80286 microprocessor. The ICE-286 emulator provides real time, full speed emulation in a users system. Popular features such as symbolic debug, 2K bytes trace memory, and single-step program execution are standard on the ICE-286 emulator. Intel provides a complete development environment using assembler (ASM-286) as well as high-level languages such as Intel's iC286, PL/M-286 or Fortran 286 to accelerate development schedules.

Intel's ICE-286 emulator is hosted on IBM's Personal Computer AT, already available as a standard development solution in most of today's engineering environments. The ICE-286 emulator operates in prototype or standalone mode allowing software development and debug before a prototype system is available. The ICE-286 emulator is ideally suited for developing real time applications such as process control, machine control, communications, or other applications requiring the full power of the 12.5 MHz 80286 microprocessor.

ICE-286 FEATURES

- Full 12.5 MHz Emulation Speed
- 2K Bytes Deep Trace Memory
- Two-Level Breakpoints with Occurrence Counters
- Single-Step Capability
- 128K Bytes Zero Wait-State Mapped Memory
- Support For Protected and Real Modes
- High-Level Language Support
- Symbolic Debug
- Numeric Processor Extension Support
- RS-232-C and GPIB Communication Links
- Crystal Power Accessory
- Interface for Intel Performance Analysis Tool (iPAT)
- Interface for Optional General Purpose Logic Analyzer
- Tutorial Software
- Complete Intel Service and Support



HIGHEST EMULATION SPEED AVAILABLE TODAY

The ICE-286 emulator supports development and debug of time-critical hardware and software using Intel's 12.5 MHz 80286 microprocessor.

RETRACE SOFTWARE TRACKS

This emulator captures up to 2048 frames of processor activity, including both execution and data bus activity. With this trace memory, large blocks of program code can be traced in real time and viewed for program flow and behavior characteristics.

HARDWARE BREAKPOINTS FOR COMPLEX DEBUG

User-defined "TIL-THEN" breakpoint statements stop emulation at specific execution addresses or bus events. During the hardware and software integration phase, breakpoint statements can be defined as execution addresses and/or bus addresses and/or bus access types, such as memory and I/O reads or writes. Additionally, event counters provide another level of breakpoint control for sophisticated state machine constructs used to specify emulation breakpoints/tracepoints.

SMALL OR LARGE STEPS

A stepping command can be used to view program execution one frame at a time or in preset frame blocks. When used in conjunction with symbolic debug, code execution can be monitored quickly and precisely.

DEBUG CODE WITHOUT A PROTOTYPE

Even before prototype hardware is available, the ICE-286 emulator working in conjunction with the Crystal Power Accessory (CPA) creates a "virtual" application environment. 128K bytes of zero wait-state memory is available for mapped memory and I/O resource addressing in 4K increments. The CPA provides emulator diagnostics as well as the ability to use the emulator without a prototype.

PROTECTED AND REAL MODES

The ICE-286 emulator has full access to all protected-mode registers and permits modification of register contents. Protected mode of execution is beneficial for secure, multitasking applications.

HIGH-LEVEL LANGUAGE SUPPORT OPTIMIZED FOR INTEL TOOLS

The ICE-286 supports emulation for programs written in Intel's ASM 286 and ASM 86 or any of the Intel high-level languages:

PL/M-286/86	Fortran-286/86
Pascal-286/86	C-286/86

These languages are optimized for Intel component architectures to deliver a tightly integrated, high performance development environment.

USER-FRIENDLY SYMBOLICS AID IN DEBUG

Symbolics allow access to program symbols by name rather than cumbersome physical addresses. Symbolic debug speeds the debugging process by reducing reliance on memory maps. In a dynamic development process, user variables can be used as parameters for ICE-286 commands resulting in a consistent debug environment.

80287 NUMERICS SUPPORT

The ICE-286 emulator provides emulation support for the 80287 numerics processor. 80287 registers can be displayed and modified allowing full debug support for numerics.

MULTIPLE HIGH-SPEED COMMUNICATION LINKS

Two communication links are available for use in conjunction with the host IBM PC AT. The ICE-286 emulator uses either serial (RS-232-C) or a parallel (GPIB) link. A user supplied National Instruments (IEEE-488) GPIB communication board provides parallel transfers at rates up to 300K bytes per second.

SOFTWARE ANALYSIS (iPAT)

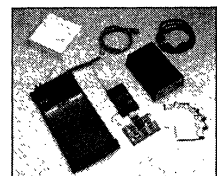
Intel's Performance Analysis Tool (iPAT) is designed to increase team productivity with features like interrupt latency measurement, code coverage analysis and software module performance analysis. These features enable the user to design reliable, high performance embedded control products. The ICE-286 emulator has an external 60 pin connector for iPAT.

BUILT-IN SUPPORT FOR LOGIC ANALYSIS

General-purpose logic analyzers can be used in conjunction with the ICE-286 to provide detailed timing of specific events. The ICE-286 emulator provides an external sync signal for triggering logic analysis, making complex trigger sequence programming easy. An additional 60 pin connector is included for the logic analyzer.

WORLDWIDE SERVICE AND SUPPORT

The ICE-286 emulator is supported by Intel's worldwide service and support organization. Total hardware and software support is available including a hotline number when the need is there.



SPECIFICATIONS

PERSONAL COMPUTER REQUIREMENTS

The ICE-286 emulator is hosted on an IBM PC AT. The emulator has been tested and evaluated on an IBM PC AT. The PC AT must meet the following minimum requirements:

- 640K Bytes of Memory
- Intel Above Board with at Least 1M Byte of Expansion Memory
- One 360K Bytes or One 1.2M Bytes Floppy Disk Drive
- One 20M Bytes Fixed-Disk Drive
- PC-DOS 3.2 or Later
- A Serial Port (COM1 or COM2) Supporting Minimally at 9600 Baud Data Transfers, or a National Instruments GPIB-PC2A Board.
- IBM PC AT BIOS

ELECTRICAL CONSIDERATIONS

Icc 1050mA

ENVIRONMENTAL SPECIFICATIONS

Temperature 10°C to 40°C Ambient
Storage Temperature -40°C to 70°C

TIMING/DC CONSIDERATIONS

ICE SM -286 USER PIN DIFFERENCES					
PARAMETER	COMPONENT SPEC.		ICE-286 SPEC.		
	Min	Max	Min	Max	
2 System (CLK) low time	11	237	14	236	
3 System (CLK) high time	13	239	14	236	
8 Read Data Setup	5		7		
10 /Ready Setup	22		24		
12a Status/peak # active delay	3	18	3	20	
12b Status/peak # inactive delay	3	20	3	22	
13 Address valid delay	1	32	1	34	
14 Write data valid delay	0	30	0	33	
15 Address/status/data float	0	32	0	34	

Consult User Guide for additional specifications

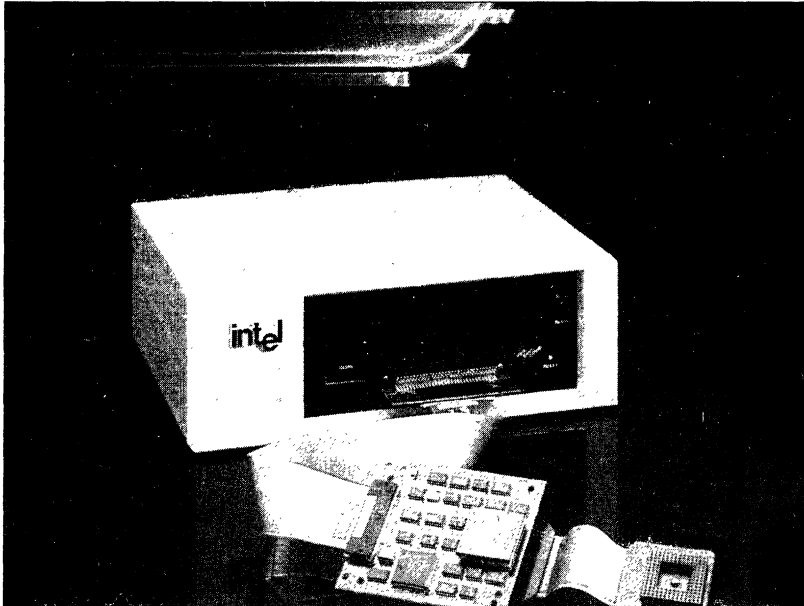
PHYSICAL DESCRIPTION AND CHARACTERISTICS

The ICE-286 Emulator consists of the following components:

Unit	Width		Height		Length	
	Inches	Cm.	Inches	Cm.	Inches	Cm.
Emulator						
Control Unit	10.40	26.40	1.70	4.30	20.70	52.60
Power Supply	7.60	19.00	4.15	10.70	11.00	27.90
User Probe	3.70	9.40	.65	1.60	7.00	17.80
User Cable/ Ploc					22.00	55.90
Hinge Cable					3.40	8.60
Crystal Power						
Accessory CPA Power	4.30	10.90	.60	1.50	6.70	17.00
Cable					9.00	22.90

ORDERING INFORMATION

- ICE286 ICE-286 NMOS System including ICE S/W packages (Requires DOS 3.XX PC AT with Above Board)
- ICE286AB ICE-286 NMOS System including ICE S/W packages and Intel's 2M Byte Above Board (PCMB 4125) (Requires DOS 3.XX PC-AT)
- ICE286PAT ICE-286 NMOS System including ICE S/W Packages and the iPAT system (Requires DOS 3.XX PC AT with Above Board)
- D86ASM286NL 286 macro assembler 286 builder/binder/mapper utilities for DOS 3.XX.
- D86C286NL 286 C compiler and run time libraries for DOS 3.XX.
- D86PLM286NL 286 PL/M compiler for DOS 3.XX.



ACCURATE AND SOPHISTICATED EMULATION FOR THE INTEL386™ FAMILY OF MICROPROCESSORS

Intel386™ In-Circuit Emulators are the cornerstone of the optimum development solution for the Intel386 family of microprocessors. From the inventor of the microprocessor comes a development tool that delivers absolute access to the sophistication of the architecture in a way that only Intel can.

Productivity boosting features such as symbolic debugging make Intel386 emulators easy to use and powerful. Intel product quality and world class technical support and service minimizes the "downtime" incurred in resolving problems. And your investment in development tools is protected via interchangeable probes for the 386™, 386SX™, and 376™ processors.

Maximize your productivity with Intel development tools. Reduced time to market and increased market acceptance for your microprocessor-based product are the benefits when Intel is the choice.

FEATURES

- Exclusive technology giving access to internal processor states provides absolutely accurate emulation history
- Unparalleled support of all of the Intel386 operating modes opens the door to the full potential of the Intel386 architecture
- Non-intrusive emulation to processor speeds of 25MHz
- Versatile event recognition makes short work of uncovering complex bugs
- Dynamic trace display of bus and execution information during emulation
- A comprehensive software development system creates the most complete development environment available from a single vendor
- A companion performance analysis tool provides analysis of software for optimized performance and reliability

FEATURES

ABSOLUTELY ACCURATE EMULATION

Intel386 Family in-circuit emulators embody exclusive technology that accesses internal processor states that are otherwise invisible. Intel386 microprocessors fetch and execute instructions in parallel; fetched instructions are not necessarily executed. Because of this, an emulator without this capability is prone to error in determining what actually occurred inside the microprocessor. With Intel's exclusive technology, an Intel386 emulator displays execution history with one hundred percent accuracy.

OPENING THE DOOR TO PROTECTED MODE

Intel386 emulators open the door to the full potential of the architecture with unparalleled support of protected mode. Not only does the emulator display and modify task state segments and global, local, and interrupt descriptor tables (with symbolic access to all descriptor components like privilege level and segment type), but emulator functions are sensitive to the operating mode of the processor, greatly improving ease of use.

Intel386 emulators support all aspects of protected mode addressing, including paged virtual memory. Processor tables are used to automatically translate virtual addresses to linear and physical addresses. Physical addresses can be translated to symbolic references to indicate the module, procedure, or data segment accessed. And when debugging a memory management system, components of the page table and directory can be displayed and modified.

FLEXIBLE AND VERSATILE EVENT RECOGNITION

Flexibility and versatility in event recognition makes short work of uncovering the most complex bugs. Bus event recognition circuitry may be used to trigger on specific or masked data input, output, read, written, or fetched at a physical address or range of addresses. Or on-chip debug registers may be used to trigger on virtual, linear, or symbolic addresses being executed, accessed, or written.

Versatility shows in other triggering options—upon a task switch, an external signal from another emulator or a logic analyzer, multiple occurrences of an event, a full trace buffer, halt or shutdown cycles, or interrupt acknowledge. And up to four sequential event triggers can be combined with a high-level construct.

Intel386 emulators continuously capture all bus activity, and optionally execution information, into a trace buffer of 4096 frames with PRE, POST, and CENTERED collection modes. The contents of the trace buffer can be displayed during full speed emulation in either execution cycle or machine-level instruction formats. Symbolic information can optionally be included in the trace display. A third trace display, the current chain of procedure calls, can be displayed when emulating high-level language programs.

SPEEDING DEVELOPMENT WITH SYMBOLICS

Intel386 processor data structures, such as registers, descriptor tables, and page tables, can be examined and modified using symbolic names. And with the symbolic debugging information that is a feature of Intel languages, memory locations can be accessed using symbolic references to the source program (such as a procedure and variable names, line numbers, or program labels) rather than via cumbersome virtual, linear, or physical addresses. The type information of variables (such as byte, word, record, or array) can also be displayed.

ACCESSING THE POWER

The power of the Intel386 emulator is reflected in the sophisticated user interface. Refined for ease of use, the command line interface contains many features to boost productivity and customize functionality.

On-line help, a syntax menu, command line editing, command history, and error message query promote ease of learning and use. I/O redirection and the ability to escape to the host operating system provide versatility for the power user. Customized procedures with variables and literal definitions can be created to assist in debugging or for manufacturing test or field service applications.

SYSTEM CONNECTIVITY AND CONFIGURATION

The Intel386 emulator can be combined with a variety of devices. I/O lines synchronize emulation starts and triggers with external tools such as a logic analyzer or another emulator. An optional time tag board synchronizes multiple Intel386 emulators and records timestamp information in the trace buffer with 20 nanosecond resolution. An optional clips pod supplements two general purpose input lines with eight data lines captured and displayed in the trace. The bus isolation board buffers the emulation processor from faults in an untested target. And with the stand-alone/self-test board the emulator can be used to debug software before the target system is functional, as well as execute confidence tests.

THE INVESTMENT PICTURE

As designs move from one Intel386 Family processor to another, the reinvestment cost is limited to probes that adapt the emulator base to the specific processor. Beside cost savings, migration from one processor to another is accomplished with minimum disruption in the engineering environment, as the same command language applies to the entire emulator family.

FEATURES

SOFTWARE COMPLETES THE SYSTEM

Intel wraps a comprehensive software development system around the emulator to deliver the most complete development environment available from a single vendor. Like the emulator, Intel's software development system supports every aspect of the Intel386 architecture.

Overlooked at times is that a significant part of developing a system is making sure the code works. Intel languages integrate seamlessly with the Intel386 emulator and provide the symbolics so important for efficient debugging. Only by using Intel languages with the Intel386 emulator can the full power of Intel development solution be utilized.

The software development system offers a broad choice of languages with object code compatibility so performance can be maximized by using different languages for specialized, performance critical modules. Architectural extensions in the high-level languages allows hardware features such as interrupts, input/output, or flags to be controlled directly, avoiding the tediousness of coding assembly language routines.

Intel's software portfolio includes a unique, sophisticated, and very powerful system builder, simplifying the generation of protected mode systems. To further reduce the effort necessary to integrate software into the final target configuration, Intel tools produce ROM-able code directly from the development system.

OPTIMIZING PERFORMANCE AND RELIABILITY

A companion performance analysis tool, iPAT™-386, provides analysis of real-time software executing on 80386-based target systems. With iPAT-386, it is possible to speed-tune applications, optimize use of operating systems, determine response characteristics, and identify code execution coverage. And iPAT-386 can be used in conjunction with an Intel386 in-circuit emulator to control test conditions.

WORLD CLASS, WORLDWIDE SERVICES

Augmenting the Intel386 Family development tools is a full array of seminars, classes and workshops; on-site consulting services; field application engineering expertise; telephone hotline support; and software and hardware maintenance contracts.

No one can match Intel's motivation to supply you with the absolute best in microprocessor development tools. When the heart of your design is an Intel microprocessor, Intel development tools help to insure we both enjoy continued success.

ORDERING INFORMATION

ICE376D	In-circuit emulator for 80376 component. Operates to 16MHz. Includes control unit, power supply, 376 processor module with PQFP adaptor, stand-alone self-test board, isolation board, and DOS host software and interface cable.	ICE38625D	In-circuit emulator for 80386 component. Operates to 25MHz. Includes control unit, power supply, 386 processor module with 132 pin PGA adaptor, stand-alone self-test board, isolation board, and DOS host software and interface cable.
ICE376DAB	Identical to ICE376D with PC/AT® compatible 2MB Above Board.	ICE38625DAB	Identical to ICE38625D with PC/AT compatible 2MB Above Board.
ICE386SXD	In-circuit emulator for 80386SX component. Operates to 16MHz. Includes control unit, power supply, 386SX processor module with PQFP adaptor, stand-alone self-test board, isolation board, and DOS host software and interface cable.	ICE376T0386SXD	Conversion kit to adapt ICE376D to support the 80386SX component. Operates to 16MHz. Includes 386SX processor module and DOS host software.
ICE386SXDAB	Identical to ICE386SXD with PC/AT compatible 2MB Above Board.	ICE376T0386D	Conversion kit to adapt ICE376D to support the 80386 component. Operates to 25MHz. Includes 386 processor module and DOS host software.

ORDERING INFORMATION

ICE386SXT0376D Conversion kit to adapt ICE386SXD to support the 80376 component. Operates to 16MHz. Includes 376 processor module and DOS host software.

For more information on Intel's Development Tools, or for the number of your nearest sales office or distributor, call 800-874-6835 (U.S.). For information or literature on additional Intel products, call 800-548-4725 (U.S. and Canada).

ICE386SXT0386D Conversion kit to adapt ICE386SXD to support the 80386 component. Operates to 25MHz. Includes 386 processor module and DOS host software.

ICE386T0376D Conversion kit to adapt ICE38625D to support the 80376 component. Operates to 16MHz. Includes 376 processor module and DOS host software.

ICE386T0386SXD Conversion kit to adapt ICE38625D to support the 80386SX component. Operates to 16MHz. Includes 386SX processor module and DOS host software.

88PGAADAPT Adaptor for ICE376D to support 88 pin PGA component packaging.

ICE3XXCPO Clips Pod Option for ICE376D, ICE386SXD, and ICE38625D.

ICE3XTTB Time Tag Board Option for ICE376D, ICE386SXD, and ICE38625D.

iPAT, Intel386, 386, 386SX, and 376 are trademarks of Intel Corporation.
PC/AT is a registered trademark of International Business Machines Corporation.

ICE™-386SX SPECIFICATIONS AND REQUIREMENTS

HOST SYSTEM REQUIREMENTS

The user supplied host system can be either an IBM® PC/AT® or Personal System/2® Model 60. Host system requirements to run the emulator include the following:

- DOS version 3.2
- 640K bytes of RAM in conventional memory
- An Above™ board with 1 megabyte of RAM configured in expanded memory mode, EMM.SYS software version 3.2
- A 20 MB hard disk
- A serial port or the National Instruments GPIB-PCII™, GPIB-PCIIA™, or MC-GPIB™ board
- A math coprocessor if either the optional time tag board is used or if a math coprocessor resides on the target system

ELECTRICAL CHARACTERISTICS

100-120V or 220-240V selectable
50-60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

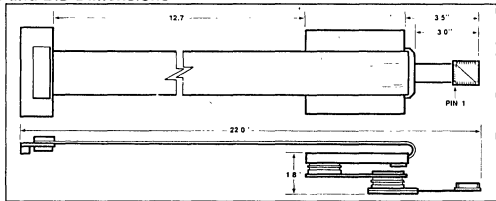
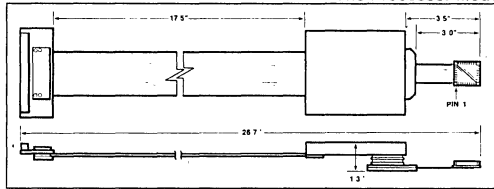
ENVIRONMENTAL CHARACTERISTICS

Operating temperature: +10 C to +40 C
(50 to 104 F)
Operating Humidity: Maximum of 85% relative humidity, non-condensing

The Emulator's Physical Characteristics

Unit	Width		Height		Length	
	inches	cm	inches	cm	inches	cm
Base Unit	13.4	34.0	4.6	11.7	11.0	27.9
Processor Module	3.8	9.7	0.7	1.8	4.4	11.2
Optional Isolation Board	3.8	9.7	0.5	1.3	4.4	11.2
Power Supply	7.7	19.6	4.1	10.4	11.0	27.9
User Cable	1.9	4.8			17.3	43.9
Target-Adapter Cable	2.3	5.3	0.5	1.3	5.1	13.0
Serial Cable					144	366
Optional Clips Pod	3.3	8.4	0.8	2.0	6.0	15.2

The Processor Module and BIB Dimensions



ELECTRICAL SPECIFICATIONS

The synchronization input lines must be valid for at least four CLK2 cycles as they are only sampled on every other cycle. These input lines are standard TTL inputs. The synchronization output lines are driven by TTL open

collector outputs that have 4.7K-ohm pull-up resistors. The synchronization input and output signals on the optional clips pod are standard TTL input and outputs.

AC Specifications With the Bus Isolation Board Installed.

Symbol	Parameter	Minimum	Maximum	Notes
t1	CLK2 period	50 nS	t1 Max	
t2a	CLK2 high time	t2a Min + 2 nS		@ 2V
t3b	CLK2 low time	t3b Min + 2 nS		@ 0.8v
t6	A1-A23 valid delay	t6 Min + 3.5 nS	t6 Max + 24.6 nS	CL = 120 pF
t7	A1-A23 float delay	t7 Min + 5.5 nS	t7 Max + 37.6 nS	
t8	BLE#, BHE# LOCK# valid delay	t8 Min + 3.5 nS	t8 Max + 24.6	CL = 75pF
t9	BLE#, BHE# LOCK# float delay	t9 Min + 5.5 nS	t9 Max + 37.6	
t10	WR#, MIO#, D/C#, ADS# valid delay	t10 Min + 3.5 nS	t10 Min + 24.6	CL = 75 pF
t11	WR#, MIO#, D/C#, ADS# float delay	t11 Min + 5.5 nS	t11 Max + 37.6	
t12	D0-D15 write data valid delay	t12 Min + 4.5 nS	t12 Max + 20.6	CL = 120 pF
t13	D0-D15 write data float delay	7.5 nS	45.6 nS	
t14	HLDA valid delay	t14 Min = 3 nS	t14 Max + 21.2 nS	
t16	NA# hold time	t16 Min + 10.6 nS		
t20	READY# hold time	t20 Min + 10.6 nS		
t21	D0-D15 read setup time	t21 Min + 8.5 nS		
t22	D0-D15 read hold time	t22 Min + 7.6 nS		
t24	HOLD hold time	t24 Min + 10.6 nS		
t25	RESET setup time	t25 Min + 2.1 nS		
t26	RESET hold time	t26 Min + 2.1 nS		
t28	NMI, INTR hold time	t28 Min + 10.6 nS		
t30	PEREQ, ERROR#, BUSY# hold time	t30 Min + 10.6 nS		



SPECIFICATIONS

Emulator Capacitance Specifications With the Target-Adapter Cable Installed

Symbol	Description	Typical (Note 1)
C_{IN}	Input Capacitance	
	CLK2	55pF
	READY#, ERROR#	35pF
	HOLD, BUSY#, PEREQ, NA#, INTR, NMI	20pF
	RESET	30pF
C_{OUT}	Output or I/O Capacitance	
	D15-D0	50pF
	A15-A1, BLE#	40pF
	A23-A16, BHE#, D/C#	30pF
	HLDA, W/R#	55pF
	ADS#, M/IO#, LOCK#	35pF

Note 1: Not tested. These specifications include the 80386SX component and all additional emulator loading.

Emulator DC Specifications Without the BIB Installed

Item	Description	Max.	Notes
$PM-I_{CC}$	Processor Module Supply Current	386SX- I_{CC} + 940 mA	
I_{IH}	Input High Leakage Current		
	A23-A1, BLE#, BHE#, D/C#, HLDA	0.02 mA	1
	D15-D0	0.06 mA	1
	ADS#, M/IO#, LOCK#, READY#, ERROR#	0.01 mA	1
	W/R#	0.03 mA	1
	CLK2	0.04 mA	1
	RESET	0.06 mA	2
I_{IL}	Input Low Leakage Current		
	A23-A1, BLE#, BHE#, D/C#	0.6 mA	1
	D15-D0	0.06 mA	1
	ADS#, M/IO#, LOCK#, READY#, ERROR#	0.01 mA	1
	W/R#	0.51 mA	1
	CLK2	0.62 mA	1
	RESET	0.6 mA	2
	HLDA	0.02 mA	1

Note 1: This specification is the DC input loading of the emulator circuitry only and does not include any 80386SX leakage current.

Note 2: This specification replaces the 80386SX specification for this signal.

Emulator DC Specifications With the BIB Installed

Item	Description	Min.	Max.
$BIB-I_{CC}$	BIB Supply Current		$PM-I_{CC}$ + 350 mA
V_{OL}	Output Low Voltage ($I_{OL} = 48$ mA)		
	A23-A1, BLE#, BHE#, D/C#, ADS#		0.5 v
	D15-D0, M/IO#, LOCK#, W/R#		0.5 v
	HLDA ($I_{OL} = 24$ mA)		0.44 v
V_{OH}	Output High Voltage ($I_{OH} = 3$ mA)		
	A23-A1, BLE#, BHE#, D/C#, ADS#	2.4 v	
	D15-D0, M/IO#, LOCK#, W/R#	2.4 v	
	HLDA ($I_{OH} = 24$ mA)	3.8 v	
I_{IH}	Input High Current		
	CLK2, RESET		1.0 μ A
	READY#		25 μ A
I_{IL}	Input Low Current		
	CLK2, RESET		1.0 μ A
	READY#		250 μ A
I_{IO}	Output Leakage Current		
	A23-A1, BLE#, BHE#, D/C#, ADS#		± 20 μ A
	D15-D0, M/IO#, LOCK#, W/R#		± 20 μ A

PROCESSOR MODULE INTERFACE CONSIDERATIONS

With the processor module directly attached to the target system without using the bus isolation board (BIB), the target system must meet the following requirements:

- The user bus controller must only drive the data bus during a valid read cycle of the emulator processor or while the emulator processor is in a hold state (the emulator processor uses the data bus to communicate with the emulator hardware).
- Before driving the address bus, the user system must gain control by asserting HOLD and receiving HLDA.
- The user reset signal is disabled during the interrogation mode. It is enabled in emulation, but is delayed by 2 or 4 CLK2 cycles.
- The user system must be able to drive one additional TTL load on all signals that go to the emulation processor.

When the target system does not satisfy the first two restrictions, the bus isolation board is used to isolate the emulation processor from the target system. With the isolation board installed, the processor CLK2 is restricted to running at 20 MHz.

The processor module derives its DC power from the target system through the 80386SX socket. It requires 1400mA, including the 80386SX current. The isolation board requires an additional 350mA.

The processor must be socketed, for example using Textool 2-0100-07243-000 or AMP 821949-4 sockets.

The printed circuit board design should locate the processor socket at the physical ends of the printed circuit board traces that connect the processor to the other logic of the target system. This reduces transmission line noise. Additionally, if the target system is enclosed in a box, pin one of the processor socket should be oriented away from the target system's box opening to make connecting the target-adaptor cable easier.

Above, ICE, Intel386, 386, 386SX, and 376 are trademarks of Intel Corporation. GPIB-PCII, GPIB-PCIIA, and MC-GPIB are trademarks of National Instruments Corporation. IBM, PC/AT, and Personal System/2 are registered trademarks of International Business Machines Corporation.

For direct information on Intel's Development Tools, or for the number of your nearest sales office or distributor, call 800-874-6835 (U.S.). For information or literature on additional Intel products, call 800-548-4725 (U.S. and Canada).

ICE™ -376 SPECIFICATIONS AND REQUIREMENTS

HOST SYSTEM REQUIREMENTS

The user supplied host system can be either an IBM® PC/AT® or Personal System/2® Model 60. Host system requirements to run the emulator include the following:

- DOS version 3.2
- 640K bytes of RAM in conventional memory
- An Above™ board with 1 megabyte of RAM configured in expanded memory mode, EMM.SYS software version 3.2
- A 20 MB hard disk
- A serial port or the National Instruments GPIB-PCII™, GPIB-PCIIA™, or MC-GPIB™ board
- A math coprocessor if either the optional time tag board is used or if a math coprocessor resides on the target system

ELECTRICAL CHARACTERISTICS

100-120V or 220-240V selectable
50-60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

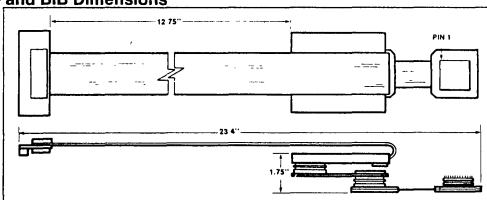
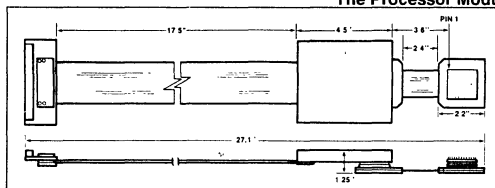
ENVIRONMENTAL CHARACTERISTICS

Operating temperature: +10 C to +40 C
(50 to 104 F)
Operating Humidity: Maximum of 85% relative humidity, non-condensing

The Emulator's Physical Characteristics

Unit	Width		Height		Length	
	inches	cm	inches	cm	inches	cm
Base Unit	13.4	34.0	4.6	11.7	11.0	27.9
Processor Module	3.8	9.7	0.7	1.8	4.4	11.2
Optional Isolation Board	3.8	9.7	0.5	1.3	4.4	11.2
Power Supply	7.7	19.6	4.1	10.4	11.0	27.9
User Cable	1.9	4.8			17.3	43.9
100-Pin Target-Adapter Cable	2.3	5.3	0.5	1.3	5.1	13.0
88-Pin Target-Adapter Cable	2.3	5.3	0.5	1.3	5.8	14.7
Serial Cable					144	366
Optional Clips Pod	3.3	8.4	0.8	2.0	6.0	15.2

The Processor Module and BIB Dimensions



ELECTRICAL SPECIFICATIONS

The synchronization input lines must be valid for at least four CLK2 cycles as they are only sampled on every other cycle. These input lines are standard TTL inputs. The synchronization output lines are driven by TTL open

collector outputs that have 4.7K-ohm pull-up resistors. The synchronization input and output signals on the optional clips pod are standard TTL input and outputs.

AC Specifications With the Bus Isolation Board Installed.

Symbol	Parameter	Minimum	Maximum	Notes
t1	CLK2 period	50 nS	t1 Max	
t2a	CLK2 high time	t2a Min+2 nS		@ 2V
t3b	CLK2 low time	t3b Min+2 nS		@ 0.8v
t6	A1-A23 float delay	t6 Min+3.5 nS	t6 Max+24.6 nS	CL= 120 pF
t7	A1-A23 float delay	t7 Min+5.5 nS	t7 Max+37.6 nS	
t8	BLE#, BHE# LOCK# valid delay	t8 Min+3.5 nS	t8 Max+24.6 nS	CL= 75pF
t9	BLE#, BHE# LOCK# float delay	t9 Min+3.5 nS	t9 Max+24.6 nS	
t10	W/R#, M/IO#, D/C#, ADS# valid delay	t10 Min+3.5 nS	t10 Max+24.6 nS	CL= 75 pF
t11	W/R#, M/IO#, D/C#, ADS# float delay	t11 Min+5.5 nS	t11 Max+37.6 nS	
t12	D0-D15 write data valid delay	t12 Min+4.5 nS	t12 Max+20.6 nS	CL= 120 pF
t13	D0-D15 write data float delay	7.5 nS	45.6 nS	
t14	HLDA valid delay	t14 Min=3 nS	t14 Max+21.2 nS	
t16	NA# hold time	t16 Min+10.6 nS		
t20	READY# hold time	t20 Min+10.6 nS		
t21	D0-D15 read setup time	t21 Min+8.5 nS		
t22	D0-D15 read hold time	t22 Min+7.6 nS		
t24	HOLD hold time	t24 Min+10.6 nS		
t25	RESET setup time	t25 Min+2.1 nS		
t26	RESET hold time	t26 Min+2.1 nS		
t28	NMI, INTR hold time	t28 Min+10.6 nS		
t30	PEREQ, ERROR#, BUSY# hold time	t30 Min+10.6 nS		

SPECIFICATIONS

Emulator Capacitance Specifications With Target-Adapter Cable Installed

Symbol	Description	Typical (Note 1)
C_{IN}	Input Capacitance	
	CLK2	55pF
	READY#, ERROR#	35pF
	HOLD, BUSY#, PEREQ, NA#, INTR, NMI	20pF
	RESET	30pF
C_{OUT}	Output or I/O Capacitance	
	D15-D0	50pF
	A15-A1, BLE#	40pF
	A23-A16, BHE#, D/C#	30pF
	HLDA, W/R#	55pF
	ADS#, M/IO#, LOCK#	35pF

Note 1: Not tested. These specifications include the 80376 component and all additional emulator loading.

Emulator DC Specifications Without the BiB Installed

Item	Description	Max.	Notes
PM- I_{CC}	Processor Module Supply Current	$376-I_{CC} + 940$ mA	
I_{IH}	Input High Leakage Current		
	A23-A1, BLE#, BHE#, D/C#, HLDA	0.02 mA	1
	D15-D0	0.06 mA	1
	ADS#, M/IO#, LOCK#, READY#, ERROR#	0.01 mA	1
	W/R#	0.03 mA	1
	CLK2	0.04 mA	1
	RESET	0.06 mA	2
I_{IL}	Input Low Leakage Current		
	A23-A1, BLE#, BHE#, D/C#	0.6 mA	1
	D15-D0	0.06 mA	1
	ADS#, M/IO#, LOCK#, READY#, ERROR#	0.01 mA	1
	W/R#	0.51 mA	1
	CLK2	0.62 mA	1
	RESET	0.6 mA	2
HLDA	0.02 mA	1	

Note 1: This specification is the DC input loading of the emulator circuitry only and does not include any 80376 leakage current.

Note 2: This specification replaces the 80376 specification for this signal.

Emulator DC Specifications With the BiB installed

Item	Description	Min.	Max.
BIB- I_{CC}	BIB Supply Current		PM- $I_{CC} + 350$ mA
V_{OL}	Output Low Voltage ($I_{OL} = 48$ mA)		0.5 v
	A23-A1, BLE#, BHE#, D/C#, ADS#		0.5 v
	D15-D0, M/IO#, LOCK#, W/R#		0.44 v
	HLDA ($I_{OL} = 24$ mA)		
V_{OH}	Output High Voltage ($I_{OH} = 3$ mA)		
	A23-A1, BLE#, BHE#, D/C#, ADS#	2.4 v	
	D15-D0, M/IO#, LOCK#, W/R#	2.4 v	
	HLDA ($I_{OH} = 24$ mA)	3.8 v	
I_{IH}	Input High Current		
	CLK2, RESET		1.0 μ A
	READY#		25 μ A
I_{IL}	Input Low Current		
	CLK2, RESET		1.0 μ A
	READY#		250 μ A
I_{IO}	Output Leakage Current		
	A23-A1, BLE#, BHE#, D/C#, ADS#		± 20 μ A
	D15-D0, M/IO#, LOCK#, W/R#		± 20 μ A

PROCESSOR MODULE INTERFACE CONSIDERATIONS

With the processor module directly attached to the target system without using the bus isolation board (BiB), the target system must meet the following requirements:

- The user bus controller must only drive the data bus during a valid read cycle of the emulator processor or while the emulator processor is in a hold state (the emulator processor uses the data bus to communicate with the emulator hardware).
- Before driving the address bus, the user system must gain control by asserting HOLD and receiving HLDA.
- The user reset signal is disabled during the interrogation mode. It is enabled in emulation, but is delayed by 2 or 4 CLK2 cycles.
- The user system must be able to drive one additional TTL load on all signals that go to the emulation processor.

When the target system does not satisfy the first two restrictions, the bus isolation board is used to isolate the emulation processor from the target system. With the isolation board installed, the processor CLK2 is restricted to running at 20 MHz.

The processor module derives its DC power from the target system through the 80376 socket. It requires 1400mA, including the 80376 current. The isolation board requires an additional 350mA.

The processor must be socketed, for example using Textool 2-0100-07243-000 or AMP 821949-4 sockets.

The printed circuit board design should locate the processor socket at the physical ends of the printed circuit board traces that connect the processor to the other logic of the target system. This reduces transmission line noise. Additionally, if the target system is enclosed in a box, pin one of the processor socket should be oriented away from the target system's box opening to make connecting the target-adaptor cable easier.

Above, ICE, Intel386, 386, 386SX, and 376 are trademarks of Intel Corporation. GPIB-PCII, GPIB-PCIIA, and MC-GPIB are trademarks of National Instruments Corporation. IBM, PC/AT, and Personal System/2 are registered trademarks of International Business Machines Corporation.

For direct information on Intel's Development Tools, or for the number of your nearest sales office or distributor, call 800-874-6835 (U.S.). For information or literature on additional Intel products, call 800-548-4725 (U.S. and Canada).

ICE™-386/25 SPECIFICATIONS AND REQUIREMENTS

HOST SYSTEM REQUIREMENTS

The user supplied host system can be either an IBM® PC/AT® or Personal System/2® Model 60. Host system requirements to run the emulator include the following:

- DOS version 3.2
- 640K bytes of RAM in conventional memory
- An Above™ board with 1 megabyte of RAM configured in expanded memory mode, EMM.SYS software version 3.2
- A 20 MB hard disk
- A serial port or the National Instruments GPIB-PCII™, GPIB-PCIIA™, or MC-GPIB™ board
- A math coprocessor if either the optional time tag board is used or if a math coprocessor resides on the target system

ELECTRICAL CHARACTERISTICS

100-120V or 220-240V selectable
50-60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

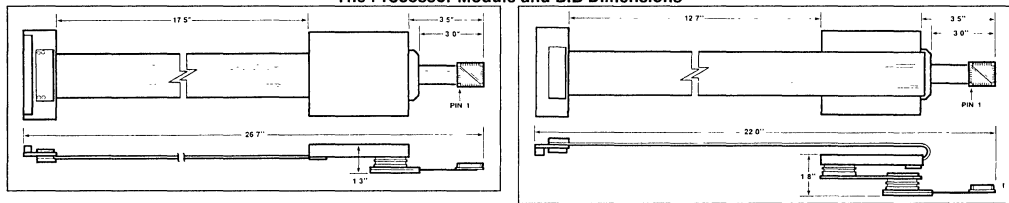
ENVIRONMENTAL CHARACTERISTICS

Operating temperature: +10 C to +40 C
(50 to 104 F)
Operating Humidity: Maximum of 85% relative humidity, non-condensing

The Emulator's Physical Characteristics

Unit	Width		Height		Length	
	inches	cm	inches	cm	inches	cm
Base Unit	13.4	34.0	4.6	11.7	11.0	27.9
Processor Module	3.8	9.7	0.7	1.8	4.4	11.2
Optional Isolation Board	3.8	9.7	0.5	1.3	4.4	11.2
Power Supply	7.7	19.6	4.1	10.4	11.0	27.9
User Cable	1.9	4.8			17.3	43.9
Target-Adapter Cable	2.3	5.3	0.5	1.3	5.8	14.7
Serial Cable					144	366
Optional Clips Pod	3.3	8.4	0.8	2.0	6.0	15.2

The Processor Module and BIB Dimensions



ELECTRICAL SPECIFICATIONS

The synchronization input lines must be valid for at least four CLK2 cycles as they are only sampled on every other cycle. These input lines are standard TTL inputs. The synchronization output lines are driven by TTL open

collector outputs that have 4.7K-ohm pull-up resistors. The synchronization input and output signals on the optional clips pod are standard TTL input and outputs.

AC Specifications With the Bus Isolation Board Installed.

Symbol	Parameter	Minimum	Maximum	Notes
t1	CLK2 period	40 nS	t1 Max	
t2a	CLK2 high time	t2a Min + 2 nS		@ 2V
t3b	CLK2 low time	t3b Min + 2 nS		@ 0.8v
t6	A2-A31 valid delay	t6 Min + 3.5 nS	16 Max + 24.6 nS	CL = 120 pF
t7	A2-A31 float delay	t7 Min + 5.5 nS	t14 Max + 32.6 nS	
t8	BE0#-BE3#, LOCK# valid delay	t8 Min + 3.5 nS	t8 Max + 24.6	CL = 75pF
t9	BE0#-BE3#, LOCK# float delay	t9 Min + 5.5 nS	t14 Min + 5.5 nS	
t10	WR#, MIO#, D/C#, ADS# valid delay	t10 Min + 3.5 nS	t10 Min + 24.6	CL = 75 pF
t11	WR#, MIO#, D/C#, ADS# float delay	t11 Min + 5.5 nS	t14 Max + 32.6	
t12	D0-D31 write data valid delay	t12 Min + 4.5 nS	t12 Max + 20.6	CL = 120 pF
t13	D0-D31 write data float delay	7.5 nS	41.6 nS	
t14	H LDA valid delay	t14 Min = 3 nS	t14 Max + 21.2 nS	
t16	NA# hold time	t16 Min + 10.6 nS		
t18	BS16# hold time	t18 Min + 10.6 nS		
t20	READY# hold time	t20 Min + 10.6 nS		
t21	D0-D31 read setup time	t21 Min + 8.5 nS		
t22	D0-D31 read hold time	t22 Min + 7.6 nS		
t24	HOLD hold time	t24 Min + 10.6 nS		
t25	RESET setup time	t25 Min + 2.1 nS		
t26	RESET hold time	t26 Min + 2.1 nS		
t28	NMI, INTR hold time	t28 Min + 10.6 nS		
t30	PEREQ, ERROR#, BUSY# hold time	t30 Min + 10.6 nS		

SPECIFICATIONS

Emulator Capacitance Specifications

Symbol	Description	Typical	TAC Installed
C _{IN}	Input Capacitance		
	CLK2	35pF	45pF
	READY#, NMI, BS16#	25pF	35pF
	HOLD, BUSY#, PEREQ, NA#, INTR, ERROR#	10pF	20pF
	RESET	20pF	30pF
C _{OUT}	Output or I/O Capacitance		
	D0-D31	40pF	50pF
	A2-A31, BE0#-BE3#	30pF	40pF
	D/C#	35pF	45pF
	W/R#	40pF	50pF
	ADS#, M/IO#, LOCK#, HLDA	25pF	35pF

Note 1: Not tested. These specifications include the 80386 component and all additional emulator loading.

Note 2: The target-adaptor cable adds a propagation delay of 0.5nS.

Emulator DC Specifications Without the BIB Installed

Item	Description	Max.	Notes
PM-I _{CC}	Processor Module Supply Current	386-I _{CC} + 1.5 A	
I _{HI}	Input High Leakage Current		1
	A2-A31, BE0#-BE3#, D0-D31 HLDA, NMI, BS16#	20μA 10μA	1 1
	ADS#, M/IO#, LOCK#, READY#	10μA	1
	W/R#, D/C#	30μA	1
	CLK2	15μA	1
	RESET	5μA	2
	I _{LI}	Input Low Leakage Current	
A2-A31, BE0#-BE3#, D0-D31 HLDA, NMI, BS16#		600μA 10μA	1 1
ADS#, M/IO#, LOCK#, READY#		10μA	1
W/R#		110μA	1
D/C#		610μA	1
CLK2		15μA	1
RESET		5μA	2

Note 1: This specification is the DC input loading of the emulator circuitry only and does not include any 80386 leakage current.

Note 2: This specification replaces the 80386 specification for this signal

Emulator DC Specifications With the BIB Installed

Item	Description	Min.	Max.
BIB-I _{CC}	BIB Supply Current		PM-I _{CC} + 475 mA
V _{OL}	Output Low Voltage (I _{OL} = 48 mA)		0.5 v
	A2-A31, BE0#-BE3#, D/C#, ADS#		0.5 v
	D0-D31, M/IO#, LOCK#, W/R#		0.44 v
	HLDA (I _{OL} = 24 mA)		
V _{OHI}	Output High Voltage (I _{OHI} = 3 mA)		
	A2-A31, BE0#-BE3#, D/C#, ADS#	2.4 v	
	D0-D31, M/IO#, LOCK#, W/R#	2.4 v	
	HLDA (I _{OHI} = 24 mA)	3.8 v	
I _{HI}	Input High Current		1.0 μA
	CLK2, RESET READY#		25 μA
I _{LI}	Input Low Current		1.0 μA
	CLK2, RESET READY#		250 μA
I _{LO}	Output Leakage Current		±20 μA
	A2-A31, BE0#-BE3#, D/C#, ADS# D0-D31, M/IO#, LOCK#, W/R#		±20 μA

PROCESSOR MODULE INTERFACE CONSIDERATIONS

With the processor module directly attached to the target system without using the bus isolation board (BIB), the target system must meet the following requirements:

- The user bus controller must only drive the data bus during a valid read cycle of the emulator processor or while the emulator processor is in a hold state (the emulator processor uses the data bus to communicate with the emulator hardware).
- Before driving the address bus, the user system must gain control by asserting HOLD and receiving HLDA.
- The user reset signal is disabled during the interrogation mode. It is enabled in emulation, but is delayed by 2 or 4 CLK2 cycles.
- The user system must be able to drive one additional TTL load on all signals that go to the emulation processor.

When the target system does not satisfy the first two restrictions, the bus isolation board is used to isolate the emulation processor from the target system. With the isolation board installed, the processor CLK2 is restricted to running at 25 MHz.

The processor module derives its DC power from the target system through the 80386 socket. It requires 1500mA, including the 80386 current. The isolation board requires an additional 475mA.

The processor must be socketed. The printed circuit board design should locate the processor socket at the physical ends of the printed circuit board traces that connect the processor to the other logic of the target system. This reduces transmission line noise. Additionally, if the target system is enclosed in a box, pin one of the processor socket should be oriented to make connecting the processor module or target-adaptor cable (TAC) easier.

The emulator uses the 386 microprocessor's pins C7, E13, and F13. The *80386 High Performance 32-Bit Microprocessor With Integrated Memory Management* data sheet specifies these pins as "N/C" (no connect). If the target system uses any of these pins, you must do one of the following:

- Use the bus isolation board.
- Use the target-adaptor cable (TAC).
- Build an adapter to disconnect pins C7, E13, and F13 (i.e., a socket with these pins removed).

Above, ICE, Intel386, 386, 386SX, and 376 are trademarks of Intel Corporation. GPIB-PCII, GPIB-PCIIA, and MC-GPIB are trademarks of National Instruments Corporation. IBM, PC/AT, and Personal System/2 are registered trademarks of International Business Machines Corporation.

For direct information on Intel's Development Tools, or for the number of your nearest sales office or distributor, call 800-874-6835 (U.S.). For information or literature on additional Intel products, call 800-548-4725 (U.S. and Canada).



DOMESTIC SALES OFFICES

ALABAMA

†Intel Corp.
5015 Bradford Dr., #2
Huntsville 35805
Tel: (205) 830-4010
FAX: (205) 837-2640

ARIZONA

†Intel Corp.
11225 N. 28th Dr.
Suite D-214
Phoenix 85029
Tel: (602) 869-4980
FAX: (602) 869-4294

Intel Corp.
1161 N. El Dorado Place
Suite 301
Tucson 85715
Tel: (602) 299-6815
FAX: (602) 296-8234

CALIFORNIA

†Intel Corp.
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel: (818) 704-8500
FAX: (818) 340-1144

†Intel Corp.
2250 E. Imperial Highway
Suite 218
El Segundo 90245
Tel: (213) 640-6040
FAX: (213) 640-7133

Intel Corp.
1510 Arden Way
Suite 101
Sacramento 95815
Tel: (916) 920-8096
FAX: (916) 920-8253

†Intel Corp.
9685 Chesapeake Dr.
Suite 325
San Diego 95123
Tel: (619) 292-8086
FAX: (619) 292-0628

†Intel Corp.*
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114
FAX: (714) 541-9157

†Intel Corp.*
San Tomas 4
2700 San Tomas Expressway
2nd Floor
Santa Clara 95051
Tel: (408) 986-8086
TWX: 910-338-0255
FAX: (408) 727-2620

COLORADO

†Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (719) 594-6622
FAX: (303) 594-0720

†Intel Corp.*
650 S. Cherry St.
Suite 915
Denver 80222
Tel: (303) 321-8096
TWX: 910-931-2289
FAX: (303) 322-8670

CONNECTICUT

†Intel Corp.
301 Lee Farm Corporate Park
83 Wooster Heights Rd.
Danbury 06810
Tel: (203) 748-3130
FAX: (203) 794-0339

FLORIDA

†Intel Corp.
6363 N.W. 6th Way
Suite 100
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407
FAX: (305) 772-8193

†Intel Corp.
5850 T.G. Lee Blvd.
Suite 340
Orlando 32822
Tel: (407) 240-8000
FAX: (407) 240-8097

†Intel Corp.
11300 4th Street North
Suite 170
St. Petersburg 33716
Tel: (813) 577-2413
FAX: (813) 578-1607

GEORGIA

†Intel Corp.
20 Technology Parkway, N.W.
Suite 150
Norcross 30092
Tel: (404) 449-0541
FAX: (404) 605-9762

ILLINOIS

†Intel Corp.*
300 N. Martingale Road
Suite 400
Schaumburg 60173
Tel: (312) 605-8031
FAX: (312) 706-9762

INDIANA

†Intel Corp.
8777 Purdue Road
Suite 125
Indianapolis 46268
Tel: (317) 875-0623
FAX: (317) 875-8938

IOWA

†Intel Corp.
1930 St. Andrews Drive N.E.
2nd Floor
Cedar Rapids 52402
Tel: (319) 393-1294

KANSAS

†Intel Corp.
10985 Cody St.
Suite 140, Bldg. D
Overland Park 66210
Tel: (913) 345-2727
FAX: (913) 345-2076

MARYLAND

†Intel Corp.*
10010 Junction Dr.
Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860
FAX: (301) 206-3677
(301) 206-3678

MASSACHUSETTS

†Intel Corp.*
Westford Corp. Center
3 Carlisle Road
2nd Floor
Westford 01886
Tel: (508) 692-3222
TWX: 710-343-6333
FAX: (508) 692-7867

MICHIGAN

†Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48322
Tel: (313) 851-8096
FAX: (313) 851-8770

MINNESOTA

†Intel Corp.
3500 W. 80th St.
Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867
FAX: (612) 831-6497

MISSOURI

†Intel Corp.
4203 Earth City Expressway
Suite 131
Earth City 63045
Tel: (614) 291-1990
FAX: (314) 291-4341

NEW JERSEY

†Intel Corp.*
Parkway 109 Office Center
328 Newman Springs Road
Red Bank 07701
Tel: (201) 747-2233
FAX: (201) 747-0983

†Intel Corp.
280 Corporate Center
75 Livingston Avenue
First Floor
Roseland 07068
Tel: (201) 740-0111
FAX: (201) 740-0626

NEW YORK

†Intel Corp.*
850 Cross Keys Office Park
Fairport 14450
Tel: (716) 425-2750
TWX: 510-253-7391
FAX: (716) 223-2561

†Intel Corp.*
2950 Expressway Dr., South
Suite 130
Islandia 11722
Tel: (516) 231-3300
TWX: 510-227-6236
FAX: (516) 348-7939

†Intel Corp.
Westage Business Center
Bldg. 300, Route 9
Fishkill 12524
Tel: (914) 897-3860
FAX: (914) 897-3125

NORTH CAROLINA

†Intel Corp.
5800 Executive Center Dr.
Suite 105
Charlotte 28212
Tel: (704) 568-8966
FAX: (704) 535-2236

†Intel Corp.
5540 Centerview Dr.
Suite 215
Raleigh 27606
Tel: (919) 851-9537
FAX: (919) 851-8974

OHIO

†Intel Corp.*
3401 Park Center Drive
Suite 220
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2529
FAX: (513) 890-8658

†Intel Corp.*
25700 Science Park Dr.
Suite 100
Beachwood 44122
Tel: (216) 464-2736
TWX: 810-427-9298
FAX: (804) 282-0673

OKLAHOMA

†Intel Corp.
6801 N. Broadway
Suite 115
Oklahoma City 73162
Tel: (405) 846-8086
FAX: (405) 840-9819

OREGON

†Intel Corp.
15254 N.W. Greenbrier Parkway
Building B
Beaverton 97005
Tel: (503) 645-8051
TWX: 910-467-8741
FAX: (503) 645-8181

PENNSYLVANIA

†Intel Corp.*
455 Pennsylvania Avenue
Suite 230
Fort Washington 19034
Tel: (215) 641-1000
TWX: 510-661-2077
FAX: (215) 641-0785

†Intel Corp.*
400 Penn Center Blvd.
Suite 610
Pittsburgh 15235
Tel: (412) 823-4970
FAX: (412) 829-7578

PUERTO RICO

†Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

†Intel Corp.
8911 Capital of Texas Hwy.
Austin 78759
Tel: (512) 794-8086
FAX: (512) 338-9335

†Intel Corp.*
12000 Ford Road
Suite 400
Dallas 75234
Tel: (214) 241-8087
FAX: (214) 484-1180

†Intel Corp.*
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490
FAX: (713) 988-3660

UTAH

†Intel Corp.
428 East 6400 South
Suite 104
Murray 84107
Tel: (801) 263-8051
FAX: (801) 268-1457

VIRGINIA

†Intel Corp.
1504 Santa Rosa Road
Suite 108
Richmond 23288
Tel: (804) 282-5689
FAX: (216) 464-2270

WASHINGTON

†Intel Corp.
155 108th Avenue N.E.
Suite 306
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002
FAX: (206) 451-9556

†Intel Corp.
408 N. Mullan Road
Suite 102
Spokane 99206
Tel: (509) 928-8086
FAX: (509) 928-9467

WISCONSIN

†Intel Corp.
330 S. Executive Dr.
Suite 102
Brookfield 53005
Tel: (414) 784-8087
FAX: (414) 796-2115

CANADA

BRITISH COLUMBIA

†Intel Semiconductor of
Canada, Ltd.
4585 Canada Way
Suite 202
Burnaby V5G 4L6
Tel: (604) 298-0387
FAX: (604) 298-8234

ONTARIO

†Intel Semiconductor of
Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Tel: (613) 829-9714
FAX: (613) 820-5936

†Intel Semiconductor of
Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
FAX: (416) 675-2438

QUEBEC

†Intel Semiconductor of
Canada, Ltd.
620 St. Jean Boulevard
Pointe Claire H9T 3K2
Tel: (514) 694-9130
FAX: 514-694-0064



DOMESTIC DISTRIBUTORS

ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35805
Tel: (205) 837-6955

†Hamilton/Avnet Electronics
4940 Research Drive
Huntsville 35805
Tel: (205) 837-7210
TWX: 810-726-2162

Pioneer/Technologies Group, Inc.
4825 University Square
Huntsville 35895
Tel: (205) 837-9300
TWX: 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 231-5140
TWX: 910-950-0077

Hamilton/Avnet Electronics
30 South McKelmy
Chandler 85226
Tel: (602) 961-6669
TWX: 910-950-0077

Arrow Electronics, Inc.
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
TWX: 910-951-1550

Wyle Distribution Group
17855 N. Black Canyon Hwy.
Phoenix 85023
Tel: (602) 249-2232
TWX: 910-951-4282

CALIFORNIA

Arrow Electronics, Inc.
10824 Hope Street
Cypress 90630
Tel: (714) 220-6300

Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (213) 701-7500
TWX: 910-483-2086

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94086
Tel: (408) 745-6600
TWX: 910-339-9371

Arrow Electronics, Inc.
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
TWX: 888-064

†Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
TWX: 910-595-2860

†Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6371
TWX: 910-595-1928

†Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
4545 Ridgeview Avenue
San Diego 92123
Tel: (619) 571-7500
TWX: 910-595-2638

†Hamilton/Avnet Electronics
9650 Desoto Avenue
Chatsworth 91311
Tel: (818) 700-1161

†Hamilton Electro Sales
10950 W. Washington Blvd.
Culver City 20230
Tel: (213) 558-2458
TWX: 910-340-6364

Hamilton Electro Sales
1361B West 190th Street
Gardena 90248
Tel: (213) 217-6700

†Hamilton/Avnet Electronics
3002 G Street
Ontario 91761
Tel: (714) 993-9411

†Avnet Electronics
20501 Plummer
Chatsworth 91351
Tel: (213) 700-6271
TWX: 910-494-2207

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4150
TWX: 910-595-2638

†Hamilton/Avnet Electronics
4103 Northgate Blvd.
Sacramento 95834
Tel: (916) 920-3150

Wyle Distribution Group
124 Maryland Street
El Segundo 90254
Tel: (213) 322-9100

Wyle Distribution Group
7382 Lampson Ave.
Garden Grove 92641
Tel: (714) 891-1717
TWX: 910-438-7140 or 7111

Wyle Distribution Group
11151 Sun Center Drive
Rancho Cordova 95670
Tel: (916) 638-5282

†Wyle Distribution Group
9525 Chocomaque Drive
San Diego 92123
Tel: (619) 565-9171
TWX: 910-335-1590

†Wyle Distribution Group
3000 Bowlers Avenue
Santa Clara 95051
Tel: (408) 727-2500
TWX: 910-338-0296

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 863-9953
TWX: 910-595-1572

Wyle Distribution Group
25677 W. Agoura Rd.
Calabasas 91302
Tel: (818) 880-9000
TWX: 910-595-2860

COLORADO

Arrow Electronics, Inc.
7060 South Tucson Way
Englewood 80112
Tel: (303) 790-4444

†Hamilton/Avnet Electronics
8765 E. Orchard Road
Suite 708
Englewood 80111
Tel: (303) 740-1017
TWX: 910-935-0787

†Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-9953
TWX: 910-936-0770

CONNECTICUT

†Arrow Electronics, Inc.
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
TWX: 710-476-0162

Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
TWX: 710-456-9974

†Pioneer Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
TWX: 710-468-3373

FLORIDA

†Arrow Electronics, Inc.
400 Fairway Drive
Suite 102
Deerfield Beach 33441
Tel: (305) 429-2800
TWX: 510-955-9456

Arrow Electronics, Inc.
37 Skyline Drive
Suite 3101
Lake Mary 32746
Tel: (407) 323-0252
TWX: 510-959-6337

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Fl. Lauderdale 33309
Tel: (305) 971-2900
TWX: 510-956-3097

†Hamilton/Avnet Electronics
3197 Tech Drive North
St. Petersburg 33702
Tel: (813) 576-3930
TWX: 810-863-0374

†Hamilton/Avnet Electronics
6947 University Boulevard
Winter Park 32792
Tel: (305) 628-3888
TWX: 810-853-0322

†Pioneer/Technologies Group, Inc.
337 S. Lake Blvd.
Alta Monte Springs 32701
Tel: (407) 834-9090
TWX: 810-653-0284

Pioneer/Technologies Group, Inc.
574 S. Military Trail
Deerfield Beach 33442
Tel: (305) 428-8877
TWX: 510-955-9653

GEORGIA

†Arrow Electronics, Inc.
3155 Northwoods Parkway
Suite A
Norcross 30071
Tel: (404) 449-8252
TWX: 810-766-0439

†Hamilton/Avnet Electronics
5825 D Peachtree Corners
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

Pioneer/Technologies Group, Inc.
3100 F Northwoods Place
Norcross 30071
Tel: (404) 448-1711
TWX: 810-766-4515

ILLINOIS

Arrow Electronics, Inc.
1140 W. Thorndale
Itasca 60143
Tel: (312) 250-0500
TWX: 312-250-0916

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (312) 860-7780
TWX: 910-227-0006

MTI Systems Sales
1100 W. Thorndale
Itasca 60143
Tel: (312) 773-2300

†Pioneer Electronics
1551 Carmen Drive
Elk Grove Village 60007
Tel: (312) 437-9680
TWX: 910-222-1834

INDIANA

†Arrow Electronics, Inc.
2495 Directors Row, Suite H
Indianapolis 46241
Tel: (317) 243-9353
TWX: 810-341-3119

Hamilton/Avnet Electronics
485 Gracie Drive
Carmel 46032
Tel: (317) 844-9333
TWX: 810-260-3966

†Pioneer Electronics
6408 Castleplace Drive
Indianapolis 46250
Tel: (317) 849-7300
TWX: 810-260-1794

IOWA

Hamilton/Avnet Electronics
915 33rd Avenue, S.W.
Cedar Rapids 52404
Tel: (319) 362-4757

KANSAS

Arrow Electronics
8208 Melrose Dr., Suite 210
Lenexa 66214
Tel: (913) 541-9542

†Hamilton/Avnet Electronics
9219 Quivera Road
Overland Park 66215
Tel: (913) 888-8900
TWX: 910-743-0005

Pioneer/Techn. Gr.
10551 Lockman Rd.
Lenexa 66215
Tel: (913) 492-0500

KENTUCKY

Hamilton/Avnet Electronics
1051 D. Newton Park
Lexington 40511
Tel: (606) 259-1475

MARYLAND

Arrow Electronics, Inc.
8300 Guilford Drive
Suite H, River Center
Columbia 21046
Tel: (301) 995-0003
TWX: 710-236-9005

Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
TWX: 710-862-1861

†Mesa Technology Corp.
9720 Pasture Woods Dr.
Columbia 21046
Tel: (301) 290-8150
TWX: 710-828-9702

†Pioneer/Technologies Group, Inc.
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 921-0660
TWX: 710-828-0545

Arrow Electronics, Inc.
7524 Standish Place
Rockville 20855
Tel: 301-424-0244

MASSACHUSETTS

Arrow Electronics, Inc.
25 Upton Dr.
Wilmington 01887
Tel: (617) 935-5134

†Hamilton/Avnet Electronics
100 Centennial Drive
Peabody 01960
Tel: (617) 531-7430
TWX: 710-393-0382

MTI Systems Sales
85 Cambridge St.
Burlington 01813

Pioneer Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 861-9200
TWX: 710-326-6617

MICHIGAN

Arrow Electronics, Inc.
755 Phoenix Drive
Ann Arbor 48104
Tel: (313) 971-8220
TWX: 810-223-6020

Hamilton/Avnet Electronics
2215 29th Street S.E.
Spacio 45
Grand Rapids 49508
Tel: (616) 243-8805
TWX: 810-274-6921

Pioneer Electronics
4504 Broadmoor S.E.
Grand Rapids 49508
FAX: 616-698-1831

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel: (313) 522-4700
TWX: 810-282-8775

†Pioneer/Michigan
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
TWX: 810-242-3271

MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
TWX: 910-576-3125

†Hamilton/Avnet Electronics
12400 Whitewater Drive
Minnetonka 55434
Tel: (612) 532-0660

†Pioneer Electronics
7625 Golden Triangle Dr.
Suite G
Eden Prairie 55343
Tel: (612) 944-3355

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63141
Tel: (314) 567-6888
TWX: 910-764-0882

†Hamilton/Avnet Electronics
13743 Shoreline Court
Earth City 63045
Tel: (314) 344-1200
TWX: 910-762-0684

NEW HAMPSHIRE

†Arrow Electronics, Inc.
3 Perimeter Road
Manchester 03103
Tel: (603) 668-6968
TWX: 710-220-1684

†Hamilton/Avnet Electronics
444 E. Industrial Drive
Manchester 03103
Tel: (603) 624-9400



DOMESTIC DISTRIBUTORS (Contd.)

NEW JERSEY

†Arrow Electronics, Inc.
Four East Stow Road
Unit 11
Marlton 08053
Tel: (609) 596-8000
TWX: 710-897-0829

†Arrow Electronics
6 Century Drive
Parsippany 07054
Tel: (201) 538-0900

†Hamilton/Avnet Electronics
1 Keystone Ave., Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
TWX: 710-940-0282

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-5300
TWX: 710-734-4388

†MTI Systems Sales
37 Kulick Rd.
Fairfield 07006
Tel: (201) 227-5552

†Pioneer Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 575-3510
TWX: 710-734-4382

NEW MEXICO

Alliance Electronics Inc.
11030 Cochiti S.E.
Albuquerque 87123
Tel: (505) 292-3360
TWX: 910-989-1151

Hamilton/Avnet Electronics
2524 Baylor Drive S.E.
Albuquerque 87106
Tel: (505) 765-1500
TWX: 910-989-0614

NEW YORK

†Arrow Electronics, Inc.
3375 Brighton Henrietta
Townline Rd.
Rochester 14623
Tel: (716) 275-0300
TWX: 510-253-4766

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11789
Tel: (516) 231-1000
TWX: 510-227-6623

Hamilton/Avnet
933 Motor Parkway
Hauppauge 11789
Tel: (516) 231-9800
TWX: 510-224-6166

†Hamilton/Avnet Electronics
333 Metro Park
Rochester 14623
Tel: (716) 475-9130
TWX: 510-253-5470

†Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13206
Tel: (315) 437-0288
TWX: 710-541-1560

†MTI Systems Sales
38 Harbor Park Drive
Fort Washington 11050
Tel: (516) 621-6200

†Pioneer Electronics
68 Corporate Drive
Binghamton 13904
Tel: (607) 722-9300
TWX: 510-252-0893

Pioneer Electronics
40 Oser Avenue
Hauppauge 11787
Tel: (516) 231-9200

†Pioneer Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
TWX: 510-221-2184

†Pioneer Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
TWX: 510-253-7001

NORTH CAROLINA

†Arrow Electronics, Inc.
5240 Greensdairy Road
Raleigh 27604
Tel: (919) 876-3132
TWX: 510-928-1856

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27604
Tel: (919) 878-0819
TWX: 510-928-1836

Pioneer/Technologies Group, Inc.
9801 A-Southern Pine Blvd.
Charlotte 28210
Tel: (919) 527-8188
TWX: 810-621-0366

OHIO

Arrow Electronics, Inc.
7620 McEwen Road
Centerville 45459
Tel: (513) 435-5563
TWX: 810-459-1611

†Arrow Electronics, Inc.
6238 Cochran Road
Solon 44139
Tel: (216) 248-3990
TWX: 810-427-9409

†Hamilton/Avnet Electronics
954 Seneca Drive
Dayton 45459
Tel: (513) 439-6733
TWX: 810-450-2531

Hamilton/Avnet Electronics
4588 Emery Industrial Pkwy.
Warrensville Heights 44128
Tel: (216) 349-5100
TWX: 810-427-9452

†Hamilton/Avnet Electronics
777 Brookside Blvd.
Westerville 43081
Tel: (614) 882-7004

†Pioneer Electronics
4433 Interpoint Boulevard
Dayton 45424
Tel: (513) 236-9900
TWX: 810-459-1622

†Pioneer Electronics
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
TWX: 810-422-2211

OKLAHOMA

Arrow Electronics, Inc.
1211 E. 51st St., Suite 101
Tulsa 74146
Tel: (918) 252-7537

†Hamilton/Avnet Electronics
12121 E. 51st St., Suite 102A
Tulsa 74146
Tel: (918) 252-7297

OREGON

†Almac Electronics Corp.
1685 N.W. 169th Place
Beaverton 97005
Tel: (503) 629-8090
TWX: 910-467-8746

†Hamilton/Avnet Electronics
6024 S.W. Jean Road
Bldg. C, Suite 10
Lake Oswego 97034
Tel: (503) 635-7848
TWX: 910-455-8179

Wyle Distribution Group
5250 N.E. Elam Young Parkway
Suite 600
Hillsboro 97124
Tel: (503) 640-6000
TWX: 910-460-2203

PENNSYLVANIA

Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000

Hamilton/Avnet Electronics
2800 Liberty Ave.
Pittsburgh 15238
Suite E
Tel: (412) 281-4150

Pioneer Electronics
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
TWX: 710-795-3122

†Pioneer/Technologies Group, Inc.
Delaware Valley
261 Gibraltar Road
Horsham 19044
Tel: (215) 674-4000
TWX: 510-665-6778

TEXAS

†Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
TWX: 910-860-5377

†Arrow Electronics, Inc.
10999 Kinghurst
Suite 100
Houston 77099
Tel: (713) 530-4700
TWX: 910-880-4439

†Arrow Electronics, Inc.
2227 W. Braker Lane
Austin 78758
Tel: (512) 835-4180
TWX: 910-874-1348

†Hamilton/Avnet Electronics
1807 W. Braker Lane
Austin 78758
Tel: (512) 837-8911
TWX: 910-874-1319

†Hamilton/Avnet Electronics
2111 W. Walnut Hill Lane
Irving 75038
Tel: (214) 550-6111
TWX: 910-860-5929

†Hamilton/Avnet Electronics
4850 Wright Rd., Suite 190
Stafford 77477
Tel: (713) 240-7733
TWX: 910-881-5523

†Pioneer Electronics
18260 Kramer
Austin 78758
Tel: (512) 835-4000
TWX: 910-874-1323

†Pioneer Electronics
13710 Omega Road
Dallas 75234
Tel: (214) 386-7300
TWX: 910-850-5563

†Pioneer Electronics
5853 Point West Drive
Houston 77036
Tel: (713) 988-5555
TWX: 910-881-1606

Wyle Distribution Group
1810 Greenville Avenue
Richardson 75081
Tel: (214) 235-9953

UTAH

Arrow Electronics
1946 Parkway Blvd.
Salt Lake City 84119
Tel: (801) 973-6913

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
TWX: 910-925-4018

Wyle Distribution Group
1325 West 2200 South
Suite E
West Valley 84119
Tel: (801) 974-9953

WASHINGTON

†Almac Electronics Corp.
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
TWX: 910-444-2067

Arrow Electronics, Inc.
19540 68th Ave. South
Kent 98032
Tel: (206) 575-4420

†Hamilton/Avnet Electronics
14212 N.E. 21st Street
Bellevue 98005
Tel: (206) 643-3950
TWX: 910-443-2469

Wyle Distribution Group
15385 N.E. 90th Street
Redmond 98052
Tel: (206) 881-1150

WISCONSIN

Arrow Electronics, Inc.
200 N. Patrick Blvd., Ste. 100
Brookfield 53005
Tel: (414) 767-6600
TWX: 910-262-1193

Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
TWX: 910-262-1182

CANADA

ALBERTA

Hamilton/Avnet Electronics
2816 21st Street N.E.
Calgary T2E 6Z3
Tel: (403) 230-3586
TWX: 09-827-642

Zentronics
Bay No. 1
3300 14th Avenue N.E.
Calgary T2A 6J4
Tel: (403) 272-1021

BRITISH COLUMBIA

†Hamilton/Avnet Electronics
105-2550 Boundary
Burnlaway V5M 3Z3
Tel: (604) 437-6667

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
TWX: 04-5077-89

MANITOBA

Zentronics
60-1313 Border Unit 60
Winnipeg R3H 0X4
Tel: (204) 694-1957

ONTARIO

Arrow Electronics, Inc.
36 Antares Dr.
Nepean K2E 7W5
Tel: (613) 226-6903

Arrow Electronics, Inc.
1093 Meyerside
Mississauga L5T 1M4
Tel: (416) 673-7769
TWX: 06-218213

†Hamilton/Avnet Electronics
6845 Rexwood Road
Units 3-4-5
Mississauga L4T 1R2
Tel: (416) 577-7432
TWX: 910-471-6867

Hamilton/Avnet Electronics
6845 Rexwood Rd., Unit 6
Mississauga L4T 1R2
Tel: (416) 277-0484

†Hamilton/Avnet Electronics
190 Colonnade Road South
Nepean K2E 7L5
Tel: (613) 226-1700
TWX: 05-349-71

†Zentronics
8 Tibury Court
Brampton L6T 3T4
Tel: (416) 451-9600
TWX: 06-976-78

†Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel: (613) 226-8840

Zentronics
60-1313 Border St.
Winnipeg R3H 0J4
Tel: (204) 694-7957

QUEBEC

†Arrow Electronics Inc.
4050 Jean Talon Ouest
Montreal H4P 1W1
Tel: (514) 735-5511
TWX: 05-25590

Arrow Electronics, Inc.
500 Avenue St-Jean Baptiste
Suite 280
Quebec G2E 5R9
Tel: (418) 871-7500
FAX: 418-871-6816

Hamilton/Avnet Electronics
2795 Halpern
St. Laurent H2E 7K1
Tel: (514) 335-1000
TWX: 610-421-3731

Zentronics
817 McCaffrey
St. Laurent H4T 1M3
Tel: (514) 737-9700
TWX: 05-827-535



EUROPEAN SALES OFFICES

DENMARK

Intel Denmark A/S
Glenvej 61, 3rd Floor
2400 Copenhagen NV
Tel: (45) (31) 19 80 33
TLX: 19567

FINLAND

Intel Finland OY
Ruusilantie 2
00390 Helsinki
Tel: (358) 0 544 644
TLX: 123232

FRANCE

Intel Corporation S.A.R.L.
1, Rue Edison-BP 303
78054 St. Quentin-en-Yvelines
Cedex
Tel: (33) (1) 30 57 70 00
TLX: 699016

WEST GERMANY

Intel Semiconductor GmbH*
Dornacher Strasse 1
8016 Feldkirchen bei Muenchen
Tel: (49) 089/90992-0
TLX: 5-23177

Intel Semiconductor GmbH
Hohenzollern Strasse 5
3000 Hannover 1
Tel: (49) 0511/344081
TLX: 9-23625

Intel Semiconductor GmbH
Abraham Lincoln Strasse 16-18
6200 Wiesbaden
Tel: (49) 06121/7605-0
TLX: 4-186183

Intel Semiconductor GmbH
Zettaching 10A
7000 Stuttgart 80
Tel: (49) 0711/7287-280
TLX: 7-254826

ISRAEL

Intel Semiconductor Ltd.*
Atidim Industrial Park-Neve Sharef
P.O. Box 43202
Tel-Aviv 61430
Tel: (972) 03-498080
TLX: 371215

ITALY

Intel Corporation Italia S.p.A.*
Milanofiori Palazzo E
20090 Assago
Milano
Tel: (39) (02) 89200950
TLX: 341286

NETHERLANDS

Intel Semiconductor B.V.*
Postbus 61430
3099 CC Rotterdam
Tel: (31) 10.407.11.11
TLX: 22283

NORWAY

Intel Norway A/S
Hvarveien 4-PO Box 92
2013 Skjetten
Tel: (47) (6) 842 420
TLX: 78018

SPAIN

Intel Iberia S.A.
Zurbaran, 28
28010 Madrid
Tel: (34) (1) 308.25.52
TLX: 46680

SWEDEN

Intel Sweden A.B.*
Dahvagen 24
171 36 Solna
Tel: (46) 8 734 01 00
TLX: 12261

SWITZERLAND

Intel Semiconductor A.G.
Zuerichstrasse
8185 Winkel-Rueti bei Zuerich
Tel: (41) 01/860 62 62
TLX: 825977

UNITED KINGDOM

Intel Corporation (U.K.) Ltd.*
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (44) (0733) 696000
TLX: 444447/8

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Electronics G.m.b.H.
Rotenmuehlgasse 26
1120 Wien
Tel: (43) (0222) 83 56 46
TLX: 31532

BELGIUM

Inelco Belgium S.A.
Av. des Croix de Guerre 94
1120 Bruxelles
Oorlogskruisenlaan, 94
1120 Brussel
Tel: (32) (02) 216 01 60
TLX: 64475 or 22090

DENMARK

ITT-Multikomponent
Naverland 29
2600 Glostrup
Tel: (45) (0) 2 45 66 45
TLX: 33 355

FINLAND

OY Fintronic AB
Melkonkatu 24A
00210 Helsinki
Tel: (358) (0) 6926022
TLX: 124224

FRANCE

Almex
Zone industrielle d'Antony
48, rue de l'Aubepine
BP 102
92164 Antony cedex
Tel: (33) (1) 46 66 21 12
TLX: 250067

Jermyn-Generim
60, rue des Gemeaux
Silic 580
94653 Runzig cedex
Tel: (33) (1) 49 78 49 78
TLX: 261585

Metrologie
Tour d'Asnieres
4, av. Laurent-Cely
92606 Asnieres Cedex
Tel: (33) (1) 47 90 62 40
TLX: 611448

Tekelec-Airtronic
Cite des Bruyeres
Rue Carle Vermet - BP 2
92310 Sevres
Tel: (33) (1) 45 34 75 35
TLX: 204552

WEST GERMANY

Electronic 2000 AG
Stahlguberring 12
8000 Muenchen 82
Tel: (49) 089/42001-0
TLX: 522561

ITT Multikomponent GmbH
Postfach 1265
Bahnhofstrasse 44
7141 Moeslingen
Tel: (49) 07141/4879
TLX: 7264472

Jermyn GmbH
Im Dachsstueck 9
6250 Limburg
Tel: (49) 06431/508-0
TLX: 415257-0

Metrologie GmbH
Meglingerstrasse 49
8000 Muenchen 71
Tel: (49) 089/78042-0
TLX: 5213189

Proelectron Vertriebs GmbH
Max Planck Strasse 1-3
6072 Dreieich
Tel: (49) 06103/30434-3
TLX: 417903

IRELAND

Micro Marketing Ltd.
Glenageary Office Park
Glenageary
Co. Dublin
Tel: (21) (353) (01) 85 63 25
TLX: 31584

ISRAEL

Eastronics Ltd.
11 Rozans Street
P.O.B. 39300
Tel-Aviv 61392
Tel: (972) 03-475151
TLX: 33638

ITALY

Intesi
Divisione ITT Industries GmbH
Viale Milanofiori
Palazzo E/5
20090 Assago (MI)
Tel: (39) 02/824701
TLX: 311351

Lasi Elettronica S.p.A.
V. le Fulvio Testi, 126
20092 Cinisello Balsamo (MI)
Tel: (39) 02/2440012
TLX: 352040

Telcom S.r.l.
Via M. Civitali 75
20148 Milano
Tel: (39) 02/4049046
TLX: 335654

ITT Multicomponents
Viale Milanofiori E/5
20090 Assago (MI)
Tel: (39) 02/824701
TLX: 311351

Silverstar
Via Dei Gracchi 20
20146 Milano
Tel: (39) 02/49961
TLX: 332189

NETHERLANDS

Koning en Hartman Elektrotechniek
B.V.
Energieweg 1
2627 AP Delft
Tel: (31) (0) 15/609906
TLX: 38250

NORWAY

Nordisk Elektronikk (Norge) A/S
Postboks 123
Smødvingen 4
1364 Hvalstad
Tel: (47) (02) 84 62 10
TLX: 77546

PORTUGAL

ATD Portugal LDA
Rua Dos Lusíadas, 5 Sala B
1300 Lisboa
Tel: (35) (1) 64 80 91
TLX: 61562

Ditram
Avenida Miguel Bombarda, 133
1000 Lisboa
Tel: (35) (1) 54 53 13
TLX: 14182

SPAIN

ATD Electronica, S.A.
Plaza Ciudad de Viena, 6
28040 Madrid
Tel: (34) (1) 234 40 00
TLX: 42477

ITT-SESA
Calle Miguel Angel, 21-3
28010 Madrid
Tel: (34) (1) 419 09 57
TLX: 27461

Metrologia Iberica, S.A.
Ctra. de Fuencarral, n.80
28100 Alcobendas (Madrid)
Tel: (34) (1) 653 86 11

SWEDEN

Nordisk Elektronik AB
Torshamnsgatan 39
Box 36
164 03 Kista
Tel: (46) 08-03 46 30
TLX: 105 47

SWITZERLAND

Industrade A.G.
Herdistrasse 31
8304 Wallisellen
Tel: (41) (01) 8328111
TLX: 56788

TURKEY

EMPA Electronic
Lindwurmstrasse 95A
8000 Muenchen 2
Tel: (49) 089/53 80 570
TLX: 528573

UNITED KINGDOM

Accent Electronic Components Ltd.
Jubilee House, Jubilee Road
Letchworth, Herts SG6 1TL
Tel: (44) (0462) 686666
TLX: 826293

Bytech-Cornway Systems
3 The Western Centre
Western Road
Bracknell RG12 1RW
Tel: (44) (0344) 55333
TLX: 847201

Jermyn
Vestry Estate
Oxford Road
Sevenoaks
Kent TN14 5EU
Tel: (44) (0732) 450144
TLX: 95142

MMD
Unit 8 Southview Park
Caversham
Reading
Berkshire RG4 0AF
Tel: (44) (0734) 481666
TLX: 846669

Rapid Silicon
Rapid House
Denmark Street
High Wycombe
Buckinghamshire HP11 2ER
Tel: (44) (0494) 442266
TLX: 937931

Rapid Systems
Rapid House
Denmark Street
High Wycombe
Buckinghamshire HP11 2ER
Tel: (44) (0494) 450244
TLX: 837931

YUGOSLAVIA

H.R. Microelectronics Corp.
2005 de la Cruz Blvd., Ste. 223
Santa Clara, CA 95050
U.S.A.
Tel: (1) (408) 988-0286
TLX: 387452

Rapido Electronic Components
S.p.a.
Via C. Beccaria, 8
34133 Trieste
Italia
Tel: (39) 040/360555
TLX: 460461



INTERNATIONAL SALES OFFICES

AUSTRALIA

Intel Australia Pty. Ltd.*
Spectrum Building
200 Pacific Hwy, Level 6
Crows Nest, NSE, 2065
Tel: 612-957-2744
FAX: 612-923-2632

BRAZIL

Intel Semicondutores do Brazil LTDA
Av. Paulista, 1159-CJS 404/405
01311 - Sao Paulo - S.P.
Tel: 55-11-287-5899
TLX: 3911153146 ISDB
FAX: 55-11-287-5119

CHINA/HONG KONG

Intel PRC Corporation
15/F, Office 1, Citic Bldg.
Jian Guo Men Wei Street
Beijing, PRC
Tel: (1) 500-4850
TLX: 22947 INTEL CN
FAX: (1) 500-2953

Intel Semiconductor Ltd.*
10/F East Tower
Bond Center
Queensway, Central
Hong Kong
Tel: (5) 8444-555
TLX: 63869 ISHLHK HX
FAX: (5) 8681-989

INDIA

Intel Asia Electronics, Inc.
4/2, Samrah Plaza
St. Mark's Road
Bangalore 560001
Tel: 011-91-812-215065
TLX: 9538452975 DCBY
FAX: 091-812-215067

JAPAN

Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26
Tel: 0298-47-8511
TLX: 3656-180
FAX: 029747-8450

Intel Japan K.K.*
Daichi Mitsugi Bldg.
1-8889 Fuchu-cho
Fuchu-shi, Tokyo 183
Tel: 0423-60-7871
FAX: 0423-60-0315

Intel Japan K.K.*
Bldg. Kumagaya
2-69 Hon-cho
Kumagaya-shi, Saitama 360
Tel: 0485-24-8871
FAX: 0485-24-7518

Intel Japan K.K.*
Mitsui-Seimei Musashi-kosugi Bldg.
915 Shinmaruko, Nakahara-ku
Kawasaki-shi, Kanagawa 211
Tel: 044-733-7011
FAX: 044-733-7010

Intel Japan K.K.
Nihon Seimei Atsugi Bldg.
1-2-1 Asahi-machi
Atsugi-shi, Kanagawa 243
Tel: 0462-29-3731
FAX: 0462-29-3781

Intel Japan K.K.*
Ryokuchi-Eki Bldg.
2-4-1 Terauchi
Toyonaka-shi, Osaka 560
Tel: 06-863-1091
FAX: 06-863-1084

Intel Japan K.K.
Shinmaru Bldg.
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100
Tel: 03-201-3621
FAX: 03-201-6850

Intel Japan K.K.
Green Bldg.
1-16-20 Nishiki
Naka-ku, Nagoya-shi
Aichi 450
Tel: 052-204-1261
FAX: 052-204-1265

KOREA

Intel Technology Asia, Ltd.
16th Floor, Life Bldg.
61 Yoito-dong, Youngdeungpo-Ku
Seoul 150-010
Tel: (2) 784-8186, 8286, 8386
TLX: K29312 INTELKO
FAX: (2) 784-8096

SINGAPORE

Intel Singapore Technology, Ltd.
101 Thomson Road #21-05/06
United Square
Singapore 1130
Tel: 250-7811
TLX: 39921 INTEL
FAX: 250-9256

TAIWAN

Intel Technology Far East Ltd.
8th Floor, No. 205
Bank Tower Bldg.
Tung Hua N. Road
Taipei
Tel: 886-2-716-9660
FAX: 886-2-717-2455

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

DAFSYS S.R.L.
Chacabuco, 90-6 PISO
1069-Buenos Aires
Tel: 54-1-334-7726
FAX: 54-1-334-1871

AUSTRALIA

Email Electronics
15-17 Hume Street
Huntingdale, 3166
Tel: 011-61-3-544-8244
TLX: AA 30895
FAX: 011-61-3-543-8179

NSD-Australia
205 Middleborough Rd.
Box Hill, Victoria 3128
Tel: 03 8900970
FAX: 03 8990819

BRAZIL

Elebra Microelectronica S.A.
Rua Geraldo Flaustina Gomes, 78
10th Floor
04575 - Sao Paulo - S.P.
Tel: 55-11-534-9641
TLX: 55-11-54593/54591
FAX: 55-11-534-9424

CHILE

DIN Instruments
Suecia 2323
Casilla 6055, Correo 22
Santiago
Tel: 56-2-225-8139
TLX: 240.846 RUD

CHINA/HONG KONG

Novel Precision Machinery Co., Ltd.
Flat D, 20 Kingsford Ind. Bldg.
Phase 1, 26 Kwai Hei Street
N.T., Kowloon
Hong Kong
Tel: 852-0-4223222
TWX: 39114 JINMI HX
FAX: 852-0-4261602

INDIA

Micronic Devices
Arun Complex
No. 65 D.V.G. Road
Basavanagudi
Bangalore 560 004
Tel: 011-91-812-600-631
011-91-812-611-365
TLX: 9538458332 MDBG

Micronic Devices
No. 516 5th Floor
Swastik Chambers
Sion, Trombay Road
Chembur
Bombay 400 071
TLX: 9531 171447 MDEV

Micronic Devices
25/8, 1st Floor
Bada Bazaar Marg
Old Rajinder Nagar
New Delhi 110 060
Tel: 011-91-11-57233509
011-91-11-589771
TLX: 031-63253 MOND IN

Micronic Devices
6-3-348/12A Dwarakapuri Colony
Hyderabad 500 482
Tel: 011-91-842-226748

S&S Corporation
1587 Kooser Road
San Jose, CA 95118
Tel: (408) 978-6216
TLX: 820281
FAX: (408) 978-8635

JAPAN

Asahi Electronics Co. Ltd.
KMM Bldg. 2-14-1 Asano
Kokurakita-ku
Kitakyushu-shi 802
Tel: 093-511-6471
FAX: 093-551-7861

C. Itoh Techno-Science Co., Ltd.
4-8-1 Dobashi, Miyamae-ku
Kawasaki-shi, Kanagawa 213
Tel: 044-852-5121
FAX: 044-877-4268

Dia Semicon Systems, Inc.
Flower Hill Shinmachi Higashi-kan
1-23-9 Shinmachi, Setagaya-ku
Tokyo 154
Tel: 03-439-1600
FAX: 03-439-1601

Okaya Koki
2-4-18 Sakae
Naka-ku, Nagoya-shi 460
Tel: 052-204-2916
FAX: 052-204-2901

Ryoyo Electro Corp.
Konwa Bldg.
1-12-22 Tsukiji
Chuo-ku, Tokyo 104
Tel: 03-546-5011
FAX: 03-546-5044

KOREA

J-Tek Corporation
6th Floor, Government Pension Bldg.
24-3 Yoito-dong
Youngdeungpo-ku
Seoul 150-010
Tel: 82-2-780-8039
TLX: 25299 KODIGIT
FAX: 82-2-784-8391

Samsung Electronics
150 Taepyungro-2 KA
Chungku, Seoul 100-102
Tel: 82-2-751-3985
TLX: 27970 KORST
FAX: 82-2-753-0967

MEXICO

SSB Electronics, Inc.
675 Palomar Street, Bldg. 4, Suite A
Chula Vista, CA 92011
Tel: (619) 585-3253
TLX: 287751 CBALL UR
FAX: (619) 585-8322

Dicopel S.A.
Tochiti 388 Fracc. Ind. San Antonio
Azcapotzalco
C.P. 02760-Mexico, D.F.
Tel: 52-5-561-3211
TLX: 177 3790 Dicome
FAX: 52-5-561-1279

PSI de Mexico
Francisco Villas Esq. Ajusto
Cuernavaca—Morelos—CEP 62130
Tel: 52-73-13-9412
FAX: 52-73-17-5333

NEW ZEALAND

Email Electronics
36 Olive Road
Penrose, Auckland
Tel: 011-64-9-591-155
FAX: 011-64-9-592-681

SINGAPORE

Electronic Resources Pte. Ltd.
17 Harvey Road #04-01
Singapore 1336
Tel: 283-8888
TWX: 56541 ERS
FAX: 2895327

SOUTH AFRICA

Electronic Building Elements
178 Erasmus Street (off Watermeyert Street)
Meyerspark, Pretoria, 0184
Tel: 011-2712-803-7680
FAX: 011-2712-803-8294

TAIWAN

Micro Electronics Corporation
5/F 587, Ming Shen East Rd.
Taipei, R.O.C.
Tel: 886-2-501-8231
TLX: 886-2-505-6609

Sertek
15/F 135, Section 2
Chien Joo North Rd.
Taipei 10479, R.O.C.
Tel: (02) 5010055
FAX: (02) 5012521
(02) 5058414

VENEZUELA

P. Benavides S.A.
Avianes a Rio
Residencia Kamarata
Locales 4 AL 7
La Candelaria, Caracas
Tel: 58-2-574-6338
TLX: 28450
FAX: 58-2-572-3321

*Field Application Location



DOMESTIC SERVICE OFFICES

ALABAMA

*Intel Corp.
5015 Bradford Dr., Suite 2
Huntsville 35805
Tel: (205) 830-4010

ALASKA

Intel Corp.
c/o TransAlaska Data Systems
300 Old Steese Hwy.
Fairbanks 99701-3120
Tel: (907) 452-4401

Intel Corp.
c/o TransAlaska Data Systems
1551 Lore Road
Anchorage 99507
Tel: (907) 522-1776

ARIZONA

*Intel Corp.
11225 N. 28th Dr.
Suite D-214
Phoenix 85029
Tel: (602) 869-4980

*Intel Corp.
500 E. Fry Blvd., Suite M-15
Sierra Vista 85635
Tel: (602) 459-5010

CALIFORNIA

†Intel Corp.
21515 Vanowen St., Ste. 116
Canoga Park 91303
Tel: (818) 704-8500

*Intel Corp.
2250 E. Imperial Hwy., Ste. 218
El Segundo 90245
Tel: (213) 640-6040

*Intel Corp.
1900 Prairie City Rd.
Folsom 95630-9597
Tel: (916) 351-6143
1-800-468-3548

Intel Corp.
9665 Cheasapeake Dr., Suite 235
San Diego 92123-1326
Tel: (619) 292-8086

**Intel Corp.
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642

**Intel Corp.
San Tomas 4
2700 San Tomas Exp., 2nd Floor
Santa Clara 95051
Tel: (408) 986-8086

COLORADO

*Intel Corp.
650 S. Cherry St., Suite 915
Denver 80222
Tel: (303) 321-8086

CONNECTICUT

*Intel Corp.
301 Lee Farm Corporate Park
83 Wooster Heights Rd.
Danbury 06810
Tel: (203) 748-3130

FLORIDA

**Intel Corp.
6363 N.W. 6th Way, Ste. 100
Ft. Lauderdale 33309
Tel: (305) 771-0600

*Intel Corp.
5850 T.G. Lee Blvd., Ste. 340
Orlando 32822
Tel: (407) 240-8000

GEORGIA

*Intel Corp.
3280 Pointe Pkwy., Ste. 200
Norcross 30092
Tel: (404) 449-0541

HAWAII

*Intel Corp.
U.S.I.S.C. Signal Batt.
Building T-1521
Shafter Plats
Shafter 96856

ILLINOIS

**Intel Corp.
300 N. Martingale Rd., Ste. 400
Schaumburg 60173
Tel: (312) 605-8031

INDIANA

*Intel Corp.
8777 Purdue Rd., Ste. 125
Indianapolis 46268
Tel: (317) 875-0623

KANSAS

*Intel Corp.
10985 Cody, Suite 140
Overland Park 66210
Tel: (913) 345-2727

MARYLAND

**Intel Corp.
10010 Junction Dr., Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860
FAX: 301-206-3677

MASSACHUSETTS

**Intel Corp.
3 Carlsle Rd., 2nd Floor
Westford 01886
Tel: (508) 692-1060

MICHIGAN

*Intel Corp.
7071 Orchard Lake Rd., Ste. 100
West Bloomfield 48322
Tel: (313) 851-8905

MINNESOTA

*Intel Corp.
3500 W. 80th St., Suite 360
Bloomington 55431
Tel: (612) 835-6722

MISSOURI

*Intel Corp.
4203 Earth City Exp., Ste. 131
Earth City 63045
Tel: (314) 291-1990

NEW JERSEY

**Intel Corp.
300 Sylvan Avenue
Englewood Cliffs 07632
Tel: (201) 567-0821

*Intel Corp.
Parkway 109 Office Center
328 Newman Springs Road
Red Bank 07701
Tel: (201) 747-2233

*Intel Corp.
280 Corporate Center
75 Livingston Ave., 1st Floor
Roseland 07068
Tel: (201) 740-0111

NEW YORK

*Intel Corp.
2950 Expressway Dr. South
Islandia 11722
Tel: (516) 231-3300

*Intel Corp.
Westage Business Center
Bldg. 300, Route 9
Fishkill 12524
Tel: (914) 897-3860

NORTH CAROLINA

*Intel Corp.
5800 Executive Dr., Ste. 105
Charlotte 28212
Tel: (704) 568-8966

**Intel Corp.
2700 Wycliff Road
Suite 102
Raleigh 27607
Tel: (919) 781-8022

OHIO

**Intel Corp.
3401 Park Center Dr., Ste. 220
Dayton 45414
Tel: (513) 890-5350

*Intel Corp.
25700 Science Park Dr., Ste. 100
Beachwood 44122
Tel: (216) 464-2736

OREGON

Intel Corp.
15254 N.W. Greenbrier Parkway
Building B
Beaverton 97005
Tel: (503) 645-8051

*Intel Corp.
5200 N.E. Elam Young Parkway
Hillsboro 97123
Tel: (503) 681-8080

PENNSYLVANIA

*Intel Corp.
455 Pennsylvania Ave., Ste. 230
Fort Washington 19034
Tel: (215) 641-1000

†Intel Corp.
400 Penn Center Blvd., Ste. 610
Pittsburgh 15235
Tel: (412) 823-4970

Intel Corp.
1513 Cedar Cliff Dr.
Camp Hill 17011
Tel: (717) 761-0860

PUERTO RICO

Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

Intel Corp.
8815 Dyer St., Suite 225
El Paso 79904
Tel: (915) 751-0186

*Intel Corp.
313 E. Anderson Lane, Suite 314
Austin 78752
Tel: (512) 454-3628

**Intel Corp.
12000 Ford Rd., Suite 401
Dallas 75234
Tel: (214) 241-8087

*Intel Corp.
7322 S.W. Freeway, Ste. 1490
Houston 77074
Tel: (713) 988-8086

UTAH

Intel Corp.
428 East 6400 South, Ste. 104
Murray 84107
Tel: (801) 263-8051

VIRGINIA

*Intel Corp.
1504 Santa Rosa Rd., Ste. 108
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

*Intel Corp.
155 108th Avenue N.E., Ste. 386
Bellevue 98004
Tel: (206) 453-8086

CANADA

ONTARIO

Intel Semiconductor of
Canada, Ltd.
2650 Queensview Dr., Ste. 250
Ottawa K2B 8H6
Tel: (613) 829-9714
FAX: 613-820-5936
Intel Semiconductor of
Canada, Ltd.
190 Athwell Dr., Ste. 102
Rexdale M9W 6H9
Tel: (416) 675-2105
FAX: 416-675-2438

CUSTOMER TRAINING CENTERS

CALIFORNIA

2700 San Tomas Expressway
Santa Clara 95051
Tel: (408) 970-1700
1-800-421-0386

ILLINOIS

300 N. Martingale Road
Suite 300
Schaumburg 60173
Tel: (708) 706-5700
1-800-421-0386

MASSACHUSETTS

3 Carlisle Road, First Floor
Westford 01886
Tel: (301) 220-3380
1-800-328-0386

MARYLAND

10010 Junction Dr.
Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860
1-800-328-0386

SYSTEMS ENGINEERING MANAGERS OFFICES

MINNESOTA

3500 W. 80th Street
Suite 360
Bloomington 55431
Tel: (612) 835-6722

NEW YORK

2950 Expressway Dr., South
Islandia 11722
Tel: (506) 231-3300

†System Engineering locations
*Carry-in locations
**Carry-in/mail-in locations



Microprocessors

Intel invented the microprocessor in 1971. Since that time we have continued our technological lead with faster and more capable products for microcomputing. And, as more and more functions are integrated on a single VLSI device, the resulting system requires less power, produces less heat and requires fewer mechanical connections – resulting in greater system reliability.

This handbook contains extensive information on the 8086, 80286 and 386™ microprocessor families as well as memory controllers, floppy disk and hard disk controllers. An entire section is devoted to development tools for the 8051, 8096, 8086/186/188, 80286 and 386 microprocessors.

Data sheets and applications notes offer many comprehensive charts, diagrams and instructions. Functional descriptions illuminate this exciting technology and make this book a “must have” for the systems designer.