



Instruction Set

6. INSTRUCTION SET

This section summarizes the 6x86MX CPU instruction set and provides detailed information on the instruction encodings.

All instructions are listed in CPU, FPU and MMX Instruction Set Summary Tables shown on pages 6-14, 6-31 and 6-38. These tables provide information on the instruction encoding, and the instruction clock counts for each instruction. The clock count values for these tables are based on the assumptions described in Section 6.3.

Depending on the instruction, the 6x86MX CPU instructions follow the general instruction format shown in Table 6-1. These instructions vary in length and can start at any byte address.

6.1 Instruction Set Format

An instruction consists of one or more bytes that can include: prefix byte(s), at least one opcode byte(s), mod r/m byte, s-i-b byte, address displacement byte(s) and immediate data byte(s). An instruction can be as short as one byte and as long as 15 bytes. If there are more than 15 bytes in the instruction a general protection fault (error code of 0) is generated.

Table 6-1. Instruction Set Format

PREFIX	OPCODE	REGISTER AND ADDRESS MODE SPECIFIER						ADDRESS DISPLACEMENT	IMMEDIATE DATA
		mod r/m Byte			s-i-b Byte				
		mod	reg	r/m	ss	Index	Base		
0 or More Bytes	1 or 2 Bytes	7 - 6	5 - 3	2 - 0	7 - 6	5 - 3	2 - 0	0, 8, 16, or 32 Bits	0, 8, 16, or 32 Bits

6.2 General Instruction Format

The fields in the general instruction format at the byte level are listed in Table 6-2.

Table 6-2. Instruction Fields

FIELD NAME		DESCRIPTION	REFERENCE
Prefix		Segment register override Address size Operand size Repeat elements in string instructions LOCK# assertion	6.2.1 (Page 6-3)
Opcode		Instruction operation	6.2.2 (Page 6-4)
mod	Address Mode Specifier	Used with r/m field to select address mode	6.2.3 (Page 6-6)
reg	General Register Specifier	Uses reg, sreg2 or sreg3 encoding depending on opcode field	6.2.4 (Page 6-7)
r/m	Address Mode Specifier	Used with mod field to select addressing mode.	6.2.3 (Page 6-6)
ss	Scale Factor	Scaled-index address mode	6.2.5 (Page 6-9)
Index		Determines general register to be selected as index register	6.2.6 (Page 6-9)
Base		Determines general register to be selected as base register	6.2.7 (Page 6-10)
Address Displacement		Determines address displacement	
Immediate data		Immediate data operand used by instruction	

6.2.1 Prefix Field

Prefix bytes can be placed in front of any instruction. The prefix modifies the operation of the next instruction only. When more than one prefix is used, the order is not important. There are five type of prefixes as follows:

1. Segment Override explicitly specifies which segment register an instruction will use for effective address calculation.
2. Address Size switches between 16- and 32-bit addressing. Selects the inverse of the default.
3. Operand Size switches between 16- and 32-bit operand size. Selects the inverse of the default.
4. Repeat is used with a string instruction which causes the instruction to be repeated for each element of the string.
5. Lock is used to assert the hardware LOCK# signal during execution of the instruction.

Table 6-3 lists the encodings for each of the available prefix bytes.

Table 6-3. Instruction Prefix Summary

PREFIX	ENCODING	DESCRIPTION
ES:	26h	Override segment default, use ES for memory operand
CS:	2Eh	Override segment default, use CS for memory operand
SS:	36h	Override segment default, use SS for memory operand
DS:	3Eh	Override segment default, use DS for memory operand
FS:	64h	Override segment default, use FS for memory operand
GS:	65h	Override segment default, use GS for memory operand
Operand Size	66h	Make operand size attribute the inverse of the default
Address Size	67h	Make address size attribute the inverse of the default
LOCK	F0h	Assert LOCK# hardware signal.
REPNE	F2h	Repeat the following string instruction.
REP/REPE	F3h	Repeat the following string instruction.

6.2.2 Opcode Field

The opcode field specifies the operation to be performed by the instruction. The opcode field is either one or two bytes in length and may be further defined by additional bits in the mod r/m byte. Some operations have more than one opcode, each specifying a different form of the operation. Some opcodes name instruction groups. For example, opcode 80h names a group of operations that have an immediate operand and a register or memory operand. The reg field may appear in the second opcode byte or in the mod r/m byte.

6.2.2.1 Opcode Field: w Bit

The 1-bit w bit (Table 6-4) selects the operand size during 16 and 32 bit data operations.

Table 6-4. w Field Encoding

w BIT	OPERAND SIZE	
	16-BIT DATA OPERATIONS	32-BIT DATA OPERATIONS
0	8 Bits	8 Bits
1	16 Bits	32 Bits

6.2.2.2 Opcode Field: d Bit

The d bit (Table 6-11) determines which operand is taken as the source operand and which operand is taken as the destination.

Table 6-5. d Field Encoding

d BIT	DIRECTION OF OPERATON	SOURCE OPERAND	DESTINATION OPERAND
0	Register --> Register or Register --> Memory	reg	mod r/m or mod ss-index-base
1	Register --> Register or Memory --> Register	mod r/m or mod ss-index-base	reg

6.2.2.3 Opcode Field: s Bit

The s bit (Table 6-11) determines the size of the immediate data field. If the S bit is set, the immediate field of the OP code is 8-bits wide and is sign extended to match the operand size of the opcode.

Table 6-6. s Field Encoding

s FIELD	IMMEDIATE FIELD SIZE		
	8-BIT OPERAND SIZE	16-BIT OPERAND SIZE	32-BIT OPERAND SIZE
0 (or not present)	8 bits	16 bits	32 bits
1	8 bits	8 bits (sign extended)	8 bits (sign extended)

6.2.2.4 Opcode Field: eee Bits

The eee field (Table 6-7) is used to select the control, debug and test registers in the MOV instructions. The type of register and base registers selected by the eee bits are listed in Table 6-7. The values shown in Table 6-7 are the only valid encodings for the eee bits.

Table 6-7. eee Field Encoding

eee BITS	REGISTER TYPE	BASE REGISTER
000	Control Register	CR0
010	Control Register	CR2
011	Control Register	CR3
100	Control Register	CR4
000	Debug Register	DR0
001	Debug Register	DR1
010	Debug Register	DR2
011	Debug Register	DR3
110	Debug Register	DR6
111	Debug Register	DR7
011	Test Register	TR3
100	Test Register	TR4
101	Test Register	TR5
110	Test Register	TR6
111	Test Register	TR7

6.2.3 mod and r/m Fields

The mod and r/m fields (Table 6-8), within the mod r/m byte, select the type of memory addressing to be used. Some instructions use a fixed addressing mode (e.g., PUSH or POP) and therefore, these fields are not present. Table 6-8 lists the addressing method when 16-bit addressing is used and a mod r/m byte is present. Some mod r/m field encodings are dependent on the w field and are shown in Table 6-9 (Page 6-7).

Table 6-8. mod r/m Field Encoding

mod and r/m fields	16-BIT ADDRESS MODE with mod r/m Byte	32-BIT ADDRESS MODE with mod r/m Byte and No s-i-b Byte Present
00 000	DS:[BX+SI]	DS:[EAX]
00 001	DS:[BX+DI]	DS:[ECX]
00 010	DS:[BP+SI]	DS:[EDX]
00 011	DS:[BP+DI]	DS:[EBX]
00 100	DS:[SI]	Note 1
00 101	DS:[DI]	DS:[d32]
00 110	DS:[d16]	DS:[ESI]
00 111	DS:[BX]	DS:[EDI]
01 000	DS:[BX+SI+d8]	DS:[EAX+d8]
01 001	DS:[BX+DI+d8]	DS:[ECX+d8]
01 010	DS:[BP+SI+d8]	DS:[EDX+d8]
01 011	DS:[BP+DI+d8]	DS:[EBX+d8]
01 100	DS:[SI+d8]	Note 1
01 101	DS:[DI+d8]	SS:[EBP+d8]
01 110	SS:[BP+d8]	DS:[ESI+d8]
01 111	DS:[BX+d8]	DS:[EDI+d8]
10 000	DS:[BX+SI+d16]	DS:[EAX+d32]
10 001	DS:[BX+DI+d16]	DS:[ECX+d32]
10 010	DS:[BP+SI+d16]	DS:[EDX+d32]
10 011	DS:[BP+DI+d16]	DS:[EBX+d32]
10 100	DS:[SI+d16]	Note 1
10 101	DS:[DI+d16]	SS:[EBP+d32]
10 110	SS:[BP+d16]	DS:[ESI+d32]
10 111	DS:[BX+d16]	DS:[EDI+d32]
11 000 through 11 111	See Table 6-9 (Page 6-7)	

Note 1: An “s-i-d” (ss, Index, Base) field is present. Refer to the ss Table 6-13 (Page 6-9), Index Table 6-14 (Page 6-9) and Base Table 6-15 (Page 6-10).

Table 6-9. mod r/m Field Encoding Dependent on w Field

mod r/m	16-BIT OPERATION w = 0	16-BIT OPERATION w = 1	32-BIT OPERATION w = 0	32-BIT OPERATION w = 1
11 000	AL	AX	AL	EAX
11 001	CL	CX	CL	ECX
11 010	DL	DX	DL	EDX
11 011	BL	BX	BL	EBX
11 100	AH	SP	AH	ESP
11 101	CH	BP	CH	EBP
11 110	DH	SI	DH	ESI
11 111	BH	DI	BH	EDI

6.2.4 reg Field

The reg field (Table 6-10) determines which general registers are to be used. The selected register is dependent on whether a 16 or 32 bit operation is current and the status of the w bit.

Table 6-10. reg Field

reg	16-BIT OPERATION w Field Not Present	32-BIT OPERATION w Field Not Present	16-BIT OPERATION w = 0	16-BIT OPERATION w = 1	32-BIT OPERATION w = 0	32-BIT OPERATION w = 1
000	AX	EAX	AL	AX	AL	EAX
001	CX	ECX	CL	CX	CL	ECX
010	DX	EDX	DL	DX	DL	EDX
011	BX	EBX	BL	BX	BL	EBX
100	SP	ESP	AH	SP	AH	ESP
101	BP	EBP	CH	BP	CH	EBP
110	SI	ESI	DH	SI	DH	ESI
111	DI	EDI	BH	DI	BH	EDI

6.2.4.1 reg Field: sreg3 Encoding

The sreg3 field (Table 6-11) is 3-bit field that is similar to the sreg2 field, but allows use of the FS and GS segment registers.

Table 6-11. sreg3 Field Encoding

sreg3 FIELD	SEGMENT REGISTER SELECTED
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	undefined
111	undefined

6.2.4.2 reg Field: sreg2 Encoding

The sreg2 field (Table 6-4) is a 2-bit field that allows one of the four 286-type segment registers to be specified.

Table 6-12. sreg2 Field Encoding

sreg2 FIELD	SEGMENT REGISTER SELECTED
00	ES
01	CS
10	SS
11	DS

6.2.5 *ss* Field

The *ss* field (Table 6-13) specifies the scale factor used in the offset mechanism for address calculation. The scale factor multiplies the index value to provide one of the components used to calculate the offset address.

Table 6-13. *ss* Field Encoding

<i>ss</i> FIELD	SCALE FACTOR
00	x1
01	x2
01	x4
11	x8

6.2.6 Index Field

The index field (Table 6-14) specifies the index register used by the offset mechanism for offset address calculation. When no index register is used (index field = 100), the *ss* value must be 00 or the effective address is undefined.

Table 6-14. Index Field Encoding

Index FIELD	INDEX REGISTER
000	EAX
001	ECX
010	EDX
011	EBX
100	none
101	EBP
110	ESI
111	EDI

6.2.7 Base Field

In Table 6-8 (Page 6-6), the note “s-i-b present” for certain entries forces the use of the mod and base field as listed in Table 6-15. The first two digits in the first column of Table 6-15 identifies the mod bits in the mod r/m byte. The last three digits in the first column of this table identifies the base fields in the s-i-b byte.

Table 6-15. mod base Field Encoding

mod FIELD WITHIN mode/rm BYTE	base FIELD WITHIN s-i-b BYTE	32-BIT ADDRESS MODE with mod r/m and s-i-b Bytes Present
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]

6.3 CPUID Instruction

The 6x86MX CPU executes the CPUID instruction (opcode 0FA2) as documented in this section only if the CPUID bit in the CCR4 configuration register is set. The CPUID instruction may be used by software to determine the vendor and type of CPU.

When the CPUID instruction is executed with EAX = 0, the ASCII characters “CyrixInstead” are placed in the EBX, EDX, and ECX registers as shown in Table 6-16:

Table 6-16. CPUID Data Returned When EAX = 0

REGISTER	CONTENTS (D31 - D0)
EBX	69 72 79 43 i r y C*
EDX	73 6E 49 78 s n l x*
ECX	64 61 65 74 d a e t*

*ASCII equivalent

When the CPUID instruction is executed with EAX = 1, EAX and EDX contain the values shown in Table 6-17.

Table 6-17. CPUID Data Returned When EAX = 1

REGISTER	CONTENTS
EAX[7 - 0]	00h
EAX[15 - 8]	06h
EDX[0]	1 = FPU Built In
EDX[1]	0 = No V86 Enhancements
EDX[2]	1 = I/O Breakpoints
EDX[3]	0 = No Page Size Extensions
EDX[4]	1 = Time Stamp Counter
EDX[5]	1 = RDMSR and WRMSR
EDX[6]	0 = No Physical Address Extensions
EDX[7]	0 = No Machine Check Exception
EDX[8]	1 = CMPXCHG8B Instruction
EDX[9]	0 = No APIC
EDX[11 - 10]	0 = Undefined
EDX[12]	0 = No Memory Type Range Registers
EDX[13]	1 = PTE Global Bit
EDX[14]	0 = No Machine Check Architecture
EDX[15]	1 = CMOV, FCMOV, FCOMI Instructions
EDX[22 - 16]	0 = Undefined
EDX[23]	1 = MMX Instructions
EDX[31 - 24]	0 = Undefined

6.4 Instruction Set Tables

The 6x86MX CPU instruction set is presented in three tables: Table 6-21. “6x86MX CPU Instruction Set Clock Count Summary” on page 6-14, Table 6-23. “6x86MX FPU Instruction Set Summary” on page 6-31 and the Table 6-25. “6x86MX Processor MMX Instruction Set Clock Count Summary” on page 6-38. Additional information concerning the FPU Instruction Set is presented on page 6-30, and the 6x86MX MMX instruction set on page 6-37.

6.4.1 Assumptions Made in Determining Instruction Clock Count

The assumptions made in determining instruction clock counts are listed below:

1. All clock counts refer to the internal CPU internal clock frequency.
2. The instruction has been prefetched, decoded and is ready for execution.
3. Bus cycles do not require wait states.
4. There are no local bus HOLD requests delaying processor access to the bus.
5. No exceptions are detected during instruction execution.
6. 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 count shown. However, if the effective address calculation uses two general register components,

add 1 clock to the clock count shown.

7. All clock counts assume aligned 32-bit memory/IO operands.
8. If instructions access a 32-bit operand that crosses a 64-bit boundary, add 1 clock for read or write and add 2 clocks for read and write.
9. For non-cached memory accesses, add two clocks (6x86MX CPU with 2x clock) or four clocks (6x86MX CPU with 3x clock). (Assumes zero wait state memory accesses).
10. Locked cycles are not cacheable. Therefore, using the LOCK prefix with an instruction adds additional clocks as specified in paragraph 9 above.
11. No parallel execution of instructions.

6.4.2 CPU Instruction Set Summary Table Abbreviations

The clock counts listed in the CPU Instruction Set Summary Table are grouped by operating mode and whether there is a register/cache hit or a cache miss. In some cases, more than one clock count is shown in a column for a given instruction, or a variable is used in the clock count. The abbreviations used for these conditions are listed in Table 6-18.

Table 6-18. CPU Clock Count Abbreviations

CLOCK COUNT SYMBOL	EXPLANATION
/	Register operand/memory operand.
n	Number of times operation is repeated.
L	Level of the stack frame.
	Conditional jump taken Conditional jump not taken. (e.g. "4 1" = 4 clocks if jump taken, 1 clock if jump not taken)
\	$CPL \leq IOPL \setminus CPL > IOPL$ (where CPL = Current Privilege Level, IOPL = I/O Privilege Level)
m	Number of parameters passed on the stack.

6.4.3 CPU Instruction Set Summary Table Flags Table

The CPU Instruction Set Summary Table lists nine flags that are affected by the execution of instructions. The conventions shown in Table 6-19 are used to identify the different flags. Table 6-20 lists the conventions used to indicate what action the instruction has on the particular flag.

Table 6-19. Flag Abbreviations

ABBREVIATION	NAME OF FLAG
OF	Overflow Flag
DF	Direction Flag
IF	Interrupt Enable Flag
TF	Trap Flag
SF	Sign Flag
ZF	Zero Flag
AF	Auxiliary Flag
PF	Parity Flag
CF	Carry Flag

Table 6-20. Action of Instruction on Flag

INSTRUCTION TABLE SYMBOL	ACTION
x	Flag is modified by the instruction.
-	Flag is not changed by the instruction.
0	Flag is reset to "0".
1	Flag is set to "1".
u	Flag is undefined following execution of the instruction.

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
AAA ASCII Adjust AL after Add	37	u - - - u u x u x	7	7		
AAD ASCII Adjust AX before Divide	D5 0A	u - - - x x u x u	7	7		
AAM ASCII Adjust AX after Multiply	D4 0A	u - - - x x u x u	13-21	13-21		
AAS ASCII Adjust AL after Subtract	3F	u - - - u u x u x	7	7		
ADC Add with Carry Register to Register Register to Memory Memory to Register Immediate to Register/Memory Immediate to Accumulator	1 [00dw] [11 reg r/m] 1 [000w] [mod reg r/m] 1 [001w] [mod reg r/m] 8 [00sw] [mod 010 r/m]### 1 [010w] ###	x - - - x x x x x	1 1 1 1 1	1 1 1 1 1	b	h
ADD Integer Add Register to Register Register to Memory Memory to Register Immediate to Register/Memory Immediate to Accumulator	0 [00dw] [11 reg r/m] 0 [000w] [mod reg r/m] 0 [001w] [mod reg r/m] 8 [00sw] [mod 000 r/m]### 0 [010w] ###	x - - - x x x x x	1 1 1 1 1	1 1 1 1 1	b	h
AND Boolean AND Register to Register Register to Memory Memory to Register Immediate to Register/Memory Immediate to Accumulator	2 [00dw] [11 reg r/m] 2 [000w] [mod reg r/m] 2 [001w] [mod reg r/m] 8 [00sw] [mod 100 r/m]### 2 [010w] ###	0 - - - x x u x 0	1 1 1 1 1	1 1 1 1 1	b	h
ARPL Adjust Requested Privilege Level From Register/Memory	63 [mod reg r/m]	- - - - - x - - -		9	a	h
BOUND Check Array Boundaries If Out of Range (Int 5) If In Range	62 [mod reg r/m]	- - - - - - - - -	20 11	20+INT 11	b, e	g,h,j,k,r
BSF Scan Bit Forward Register, Register/Memory	0F BC [mod reg r/m]	- - - - - x - - -	3	3	b	h
BSR Scan Bit Reverse Register, Register/Memory	0F BD [mod reg r/m]	- - - - - x - - -	3	3	b	h
BSWAP Byte Swap	0F C[1 reg]	- - - - - - - - -	4	4		
BT Test Bit Register/Memory, Immediate Register/Memory, Register	0F BA [mod 100 r/m]# 0F A3 [mod reg r/m]	- - - - - - - - x	2 5/6	2 5/6	b	h
BTC Test Bit and Complement Register/Memory, Immediate Register/Memory, Register	0F BA [mod 111 r/m]# 0F BB [mod reg r/m]	- - - - - - - - x	3 5/6	3 5/6	b	h

= immediate 8-bit data
= immediate 16-bit data
= full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
BTR <i>Test Bit and Reset</i> Register/Memory, Immediate Register/Memory, Register	0F BA [mod 110 r/m]# 0F B3 [mod reg r/m]	- - - - - - - - x	3 5/6	3 5/6	b	h
BTS <i>Test Bit and Set</i> Register/Memory Register (short form)	0F BA [mod 101 r/m] 0F AB [mod reg r/m]	- - - - - - - - x	3 5/6	3 5/6	b	h
CALL <i>Subroutine Call</i> Direct Within Segment Register/Memory Indirect Within Segment Direct Intersegment Call Gate to Same Privilege Call Gate to Different Privilege No Parameters Call Gate to Different Privilege m Par's 16-bit Task to 16-bit TSS 16-bit Task to 32-bit TSS 16-bit Task to V86 Task 32-bit Task to 16-bit TSS 32-bit Task to 32-bit TSS 32-bit Task to V86 Task Indirect Intersegment Call Gate to Same Privilege Call Gate to Different Privilege No Parameters Call Gate to Different Privilege Level m Par's 16-bit Task to 16-bit TSS 16-bit Task to 32-bit TSS 16-bit Task to V86 Task 32-bit Task to 16-bit TSS 32-bit Task to 32-bit TSS 32-bit Task to V86 Task	E8 +++ FF [mod 010 r/m] 9A [unsigned full offset, selector] FF [mod 011 r/m]	- - - - - - - - -	1 1/3 3 5	1 1/3 4 15 26 35+2m 110 118 96 112 120 98 8 20 31 40+2m 114 122 100 116 124 102	b	h,j,k,r
CBW <i>Convert Byte to Word</i>	98	- - - - - - - - -	3	3		
CDQ <i>Convert Doubleword to Quadword</i>	99	- - - - - - - - -	2	2		
CLC <i>Clear Carry Flag</i>	F8	- - - - - - - - 0	1	1		
CLD <i>Clear Direction Flag</i>	FC	- 0 - - - - - - -	7	7		
CLI <i>Clear Interrupt Flag</i>	FA	- - 0 - - - - - - -	7	7		m
CLTS <i>Clear Task Switched Flag</i>	0F 06	- - - - - - - - -	10	10	c	l
CMC <i>Complement the Carry Flag</i>	F5	- - - - - - - - x	2	2		

= immediate 8-bit data
= immediate 16-bit data
= full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined

PRELIMINARY

6-15

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
CMP <i>Compare Integers</i> Register to Register Register to Memory Memory to Register Immediate to Register/Memory Immediate to Accumulator	3 [10dw] [11 reg r/m] 3 [101w] [mod reg r/m] 3 [100w] [mod reg r/m] 8 [00sw] [mod 111 r/m] ### 3 [110w] ###	x - - - x x x x x	1 1 1 1 1	1 1 1 1 1	b	h
CMOVA/CMOVNB <i>Move if Above/ Not Below or Equal</i> Register, Register/Memory	0F 47 [mod reg r/m]	- - - - -	1	1		r
CMOVBE/CMOVNA <i>Move if Below or Equal/ Not Above</i> Register, Register/Memory	0F 46 [mod reg r/m]	- - - - -	1	1		r
CMOVAE/CMOVNB/CMOVNC/ <i>Move if Above or Equal/Not Below/Not Carry</i> Register, Register/Memory	0F 43 [mod reg r/m]	- - - - -	1	1		r
CMOVB/CMOVNC/CMOVNAE <i>Move if Below/ Carry/Not Above or Equal</i> Register, Register/Memory	0F 42 [mod reg r/m]	- - - - -	1	1		r
CMOVE/CMOVZ <i>Move if Equal/Zero</i> Register, Register/Memory	0F 44 [mod reg r/m]	- - - - -	1	1		r
CMOVNE/CMOVNZ <i>Move if Not Equal/ Not Zero</i> Register, Register/Memory	0F 45 [mod reg r/m]	- - - - -	1	1		r
CMOVG/CMOVNLE <i>Move if Greater/ Not Less or Equal</i> Register, Register/Memory	0F 4F [mod reg r/m]	- - - - -	1	1		r
CMOVLE/CMOVNG <i>Move if Less or Equal/ Not Greater</i> Register, Register/Memory	0F 4E [mod reg r/m]	- - - - -	1	1		r
CMOVL/CMOVNGE <i>Move if Less/ Not Greater or Equal</i> Register, Register/Memory	0F 4C [mod reg r/m]	- - - - -	1	1		r
CMOVGE/CMOVNL <i>Move if Greater or Equal/ Not Less</i> Register, Register/Memory	0F 4D [mod reg r/m]	- - - - -	1	1		r

= immediate 8-bit data
= immediate 16-bit data
= full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
CMOVO <i>Move if Overflow</i> Register, Register/Memory	0F 40 [mod reg r/m]	- - - - - - - -	1	1		r
CMOVNO <i>Move if No Overflow</i> Register, Register/Memory	0F 41 [mod reg r/m]	- - - - - - - -	1	1		r
CMOVP/CMOVPE <i>Move if Parity/Parity Even</i> Register, Register/Memory	0F 4A [mod reg r/m]	- - - - - - - -	1	1		r
CMONP/CMOVPO <i>Move if Not Parity/ Parity Odd</i> Register, Register/Memory	0F 4B [mod reg r/m]	- - - - - - - -	1	1		r
CMOVS <i>Move if Sign</i> Register, Register/Memory	0F 48 [mod reg r/m]	- - - - - - - -	1	1		r
CMOVNS <i>Move if Not Sign</i> Register, Register/Memory	0F 49 [mod reg r/m]	- - - - - - - -	1	1		r
CMPS <i>Compare String</i>	A [011w]	x - - - x x x x x	5	5	b	h
CMPXCHG <i>Compare and Exchange</i> Register1, Register2 Memory, Register	0F B [000w] [11 reg2 reg1] 0F B [000w] [mod reg r/m]	x - - - x x x x x	11 11	11 11		
CMPXCHG8B <i>Compare and Exchange 8 Bytes</i>	0F C7 [mod 001 r/m]	- - - - - - - -				
CPUID <i>CPU Identification</i>	0F A2	- - - - - - - -	12	12		
CWD <i>Convert Word to Doubleword</i>	99	- - - - - - - -	2	2		
CWDE <i>Convert Word to Doubleword Extended</i>	98	- - - - - - - -	2	2		
DAA <i>Decimal Adjust AL after Add</i>	27	- - - - x x x x x	9	9		
DAS <i>Decimal Adjust AL after Subtract</i>	2F	- - - - x x x x x	9	9		
DEC <i>Decrement by 1</i> Register/Memory Register (short form)	F [111w] [mod 001 r/m] 4 [1 reg]	x - - - x x x x -	1 1	1 1	b	h
DIV <i>Unsigned Divide</i> Accumulator by Register/Memory Divisor: Byte Word Doubleword	F [011w] [mod 110 r/m]	- - - - x x u u -	13-17 13-25 13-41	13-17 13-25 13-41	b,e	e,h

= immediate 8-bit data
= immediate 16-bit data
= full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
ENTER Enter New Stack Frame Level = 0 Level = 1 Level (L) > 1	C8 ##, #	- - - - - - - - - -	10 13 10+L*3	10 13 10+L*3	b	h
HLT Halt	F4	- - - - - - - - - -	5	5		l
IDIV Integer (Signed) Divide Accumulator by Register/Memory Divisor: Byte Word Doubleword	F [011w] [mod 111 r/m]	- - - - x x u u -	16-20 16-28 17-45	16-20 16-28 17-45	b,e	e,h
IMUL Integer (Signed) Multiply Accumulator by Register/Memory Multiplier: Byte Word Doubleword Register with Register/Memory Multiplier: Word Doubleword Register/Memory with Immediate to Register2 Multiplier: Word Doubleword	F [011w] [mod 101 r/m] 0F AF [mod reg r/m] 6 [10s1] [mod reg r/m] ###	x - - - x x u u x	4 4 10 4 10 5 11	4 4 10 4 10 5 11	b	h
IN Input from I/O Port Fixed Port Variable Port	E [010w] [#] E [110w]	- - - - - - - - - -	14 14	14/28 14/28		m
INC Increment by 1 Register/Memory Register (short form)	F [111w] [mod 000 r/m] 4 [0 reg]	x - - - x x x x -	1 1	1 1	b	h
INS Input String from I/O Port	6 [110w]	- - - - - - - - - -	14	14/28	b	h,m

= immediate 8-bit data
= immediate 16-bit data
= full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/Cache Hit	Reg/Cache Hit	Real Mode	Protected Mode
INT <i>Software Interrupt</i> INT i Protected Mode: Interrupt or Trap to Same Privilege Interrupt or Trap to Different Privilege 16-bit Task to 16-bit TSS by Task Gate 16-bit Task to 32-bit TSS by Task Gate 16-bit Task to V86 by Task Gate 16-bit Task to 16-bit TSS by Task Gate 32-bit Task to 32-bit TSS by Task Gate 32-bit Task to V86 by Task Gate V86 to 16-bit TSS by Task Gate V86 to 32-bit TSS by Task Gate V86 to Privilege 0 by Trap Gate/Int Gate INT 3 INTO If OF==0 If OF==1 (INT 4)	CD #	- - x 0 - - - - -	9		b,e	g,j,k,r
				21		
				32		
				114		
				122		
				100		
				116		
				124		
				102		
				124		
			102			
			46			
	CC		INT	INT		
	CE		6	6		
				15+INT		
INVD <i>Invalidate Cache</i>	0F 08	- - - - - - - - -	12	12	t	t
INVLPG <i>Invalidate TLB Entry</i>	0F 01 [mod 111 r/m]	- - - - - - - - -	13	13		
IRET <i>Interrupt Return</i> Real Mode Protected Mode: Within Task to Same Privilege Within Task to Different Privilege 16-bit Task to 16-bit Task 16-bit Task to 32-bit TSS 16-bit Task to V86 Task 32-bit Task to 16-bit TSS 32-bit Task to 32-bit TSS 32-bit Task to V86 Task	CF	x x x x x x x x x	7			g,h,j,k,r
				10		
				26		
				117		
				125		
				103		
				119		
				127		
				105		
JB/JNAE/JC <i>Jump on Below/Not Above or Equal/Carry</i> 8-bit Displacement Full Displacement	72 +	- - - - - - - - -	1	1		r
	0F 82 +++		1	1		
JBE/JNA <i>Jump on Below or Equal/Not Above</i> 8-bit Displacement Full Displacement	76 +	- - - - - - - - -	1	1		r
	0F 86 +++		1	1		

= immediate 8-bit data
 ## = immediate 16-bit data
 ### = full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
 +++ = full signed displacement (16, 32 bits)

x = modified
 - = unchanged
 u = undefined

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS								REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES		
		OF	DF	IF	TF	SF	ZF	AF	PF	CF	Reg/Cache Hit	Reg/Cache Hit	Real Mode	Protected Mode
JCXZ/JECXZ <i>Jump on CX/ECX Zero</i>	E3 +	-	-	-	-	-	-	-	-	-	1	1		r
JE/JZ <i>Jump on Equal/Zero</i> 8-bit Displacement Full Displacement	74 + 0F 84 +++	-	-	-	-	-	-	-	-	-	1 1	1 1		r
JL/JNGE <i>Jump on Less/Not Greater or Equal</i> 8-bit Displacement Full Displacement	7C + 0F 8C +++	-	-	-	-	-	-	-	-	-	1 1	1 1		r
JLE/JNG <i>Jump on Less or Equal/Not Greater</i> 8-bit Displacement Full Displacement	7E + 0F 8E +++	-	-	-	-	-	-	-	-	-	1 1	1 1		r
JMP <i>Unconditional Jump</i> 8-bit Displacement Full Displacement Register/Memory Indirect Within Segment Direct Intersegment Call Gate Same Privilege Level 16-bit Task to 16-bit TSS 16-bit Task to 32-bit TSS 16-bit Task to V86 Task 32-bit Task to 16-bit TSS 32-bit Task to 32-bit TSS 32-bit Task to V86 Task Indirect Intersegment Call Gate Same Privilege Level 16-bit Task to 16-bit TSS 16-bit Task to 32-bit TSS 16-bit Task to V86 Task 32-bit Task to 16-bit TSS 32-bit Task to 32-bit TSS 32-bit Task to V86 Task	EB + E9 +++ FF [mod 100 r/m] EA [unsigned full offset, selector] FF [mod 101 r/m]	-	-	-	-	-	-	-	-	-	1 1 1/3 1	1 1 1/3 4	b	h,j,k,r
JNB/JAE/JNC <i>Jump on Not Below/Above or Equal/Not Carry</i> 8-bit Displacement Full Displacement	73 + 0F 83 +++	-	-	-	-	-	-	-	-	-	1 1	1 1		r
JNBE/JA <i>Jump on Not Below or Equal/Above</i> 8-bit Displacement Full Displacement	77 + 0F 87 +++	-	-	-	-	-	-	-	-	-	1 1	1 1		r
JNE/JNZ <i>Jump on Not Equal/Not Zero</i> 8-bit Displacement Full Displacement	75 + 0F 85 +++	-	-	-	-	-	-	-	-	-	1 1	1 1		r

= immediate 8-bit data
= immediate 16-bit data
= full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined



Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
JNL/JGE <i>Jump on Not Less/Greater or Equal</i> 8-bit Displacement Full Displacement	7D + 0F 8D +++	- - - - - - - -	1 1	1 1		r
JNLE/JG <i>Jump on Not Less or Equal/Greater</i> 8-bit Displacement Full Displacement	7F + 0F 8F +++	- - - - - - - -	1 1	1 1		r
JNO <i>Jump on Not Overflow</i> 8-bit Displacement Full Displacement	71 + 0F 81 +++	- - - - - - - -	1 1	1 1		r
JNP/JPO <i>Jump on Not Parity/Parity Odd</i> 8-bit Displacement Full Displacement	7B + 0F 8B +++	- - - - - - - -	1 1	1 1		r
JNS <i>Jump on Not Sign</i> 8-bit Displacement Full Displacement	79 + 0F 89 +++	- - - - - - - -	1 1	1 1		r
JO <i>Jump on Overflow</i> 8-bit Displacement Full Displacement	70 + 0F 80 +++	- - - - - - - -	1 1	1 1		r
JP/JPE <i>Jump on Parity/Parity Even</i> 8-bit Displacement Full Displacement	7A + 0F 8A +++	- - - - - - - -	1 1	1 1		r
JS <i>Jump on Sign</i> 8-bit Displacement Full Displacement	78 + 0F 88 +++	- - - - - - - -	1 1	1 1		r
LAHF <i>Load AH with Flags</i>	9F	- - - - - - - -	2	2		
LAR <i>Load Access Rights From Register/Memory</i>	0F 02 [mod reg r/m]	- - - - - x - - -		8	a	g,h,j,p
LDS <i>Load Pointer to DS</i>	C5 [mod reg r/m]	- - - - - - - -	2	4	b	h,i,j
LEA <i>Load Effective Address</i> No Index Register With Index Register	8D [mod reg r/m]	- - - - - - - -	1 1	1 1		
LEAVE <i>Leave Current Stack Frame</i>	C9	- - - - - - - -	4	4	b	h
LES <i>Load Pointer to ES</i>	C4 [mod reg r/m]	- - - - - - - -	2	4	b	h,i,j

= immediate 8-bit data
= immediate 16-bit data
= full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
LFS Load Pointer to FS	0F B4 [mod reg r/m]	- - - - - - - -	2	4	b	h,i,j
LGDT Load GDT Register	0F 01 [mod 010 r/m]	- - - - - - - -	8	8	b,c	h,l
LGS Load Pointer to GS	0F B5 [mod reg r/m]	- - - - - - - -	2	4	b	h,i,j
LIDT Load IDT Register	0F 01 [mod 011 r/m]	- - - - - - - -	8	8	b,c	h,l
LLDT Load LDT Register From Register/Memory	0F 00 [mod 010 r/m]	- - - - - - - -	5	5	a	g,h,j,l
LMSW Load Machine Status Word From Register/Memory	0F 01 [mod 110 r/m]	- - - - - - - -	13	13	b,c	h,l
LODS Load String	A [110 w]	- - - - - - - -	3	3	b	h
LOOP Offset Loop/No Loop	E2 +	- - - - - - - -	1	1		r
LOOPNZ/LOOPNE Offset	E0 +	- - - - - - - -	1	1		r
LOOPZ/LOOPE Offset	E1 +	- - - - - - - -	1	1		r
LSL Load Segment Limit From Register/Memory	0F 03 [mod reg r/m]	- - - - - x - -		8	a	g,h,j,p
LSS Load Pointer to SS	0F B2 [mod reg r/m]	- - - - - - - -	2	4	a	h,i,j
LTR Load Task Register From Register/Memory	0F 00 [mod 011 r/m]	- - - - - - - -		7	a	g,h,j,l
MOV Move Data Register to Register	8 [10dw] [11 reg r/m]	- - - - - - - -	1	1	b	h,i,j
Register to Memory	8 [100w] [mod reg r/m]		1	1		
Register/Memory to Register	8 [101w] [mod reg r/m]		1	1		
Immediate to Register/Memory	C [011w] [mod 000 r/m] ###		1	1		
Immediate to Register (short form)	B [w reg] ###		1	1		
Memory to Accumulator (short form)	A [000w] +++		1	1		
Accumulator to Memory (short form)	A [001w] +++		1	1		
Register/Memory to Segment Register	8E [mod sreg3 r/m]		1	1/3		
Segment Register to Register/Memory	8C [mod sreg3 r/m]		1	1		
MOV Move to/from Control/Debug/Test Regs		- - - - - - - -				1
Register to CR0/CR2/CR3/CR4	0F 22 [11 eee reg]		20/5/5	20/5/5		
CR0/CR2/CR3/CR4 to Register	0F 20 [11 eee reg]		6	6		
Register to DR0-DR3	0F 23 [11 eee reg]		16	16		
DR0-DR3 to Register	0F 21 [11 eee reg]		14	14		
Register to DR6-DR7	0F 23 [11 eee reg]		16	16		
DR6-DR7 to Register	0F 21 [11 eee reg]		14	14		
Register to TR3-5	0F 26 [11 eee reg]		10	10		
TR3-5 to Register	0F 24 [11 eee reg]		5	5		
Register to TR6-TR7	0F 26 [11 eee reg]		10	10		
TR6-TR7 to Register	0F 24 [11 eee reg]		6	6		

= immediate 8-bit data
 ## = immediate 16-bit data
 ### = full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
 +++ = full signed displacement (16, 32 bits)

x = modified
 - = unchanged
 u = undefined

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/Cache Hit	Reg/Cache Hit	Real Mode	Protected Mode
MOVS <i>Move String</i>	A [010w]	- - - - - - - -	4	4	b	h
MOVSBX <i>Move with Sign Extension Register from Register/Memory</i>	0F B[111w] [mod reg r/m]	- - - - - - - -	1	1	b	h
MOVZXB <i>Move with Zero Extension Register from Register/Memory</i>	0F B[011w] [mod reg r/m]	- - - - - - - -	1	1	b	h
MUL <i>Unsigned Multiply Accumulator with Register/Memory</i> Multiplier: Byte Word Doubleword	F [011w] [mod 100 r/m]	x - - - x x u u x	4 4 10	4 4 10	b	h
NEG <i>Negate Integer</i>	F [011w] [mod 011 r/m]	x - - - x x x x x	1	1	b	h
NOP <i>No Operation</i>	90	- - - - - - - -	1	1		
NOT <i>Boolean Complement</i>	F [011w] [mod 010 r/m]	- - - - - - - -	1	1	b	h
OIO <i>Official Invalid OpCode</i>	0F FF	- - x 0 - - - - -	1	8 - 125		
OR <i>Boolean OR</i> Register to Register Register to Memory Memory to Register Immediate to Register/Memory Immediate to Accumulator	0 [10dw] [11 reg r/m] 0 [100w] [mod reg r/m] 0 [101w] [mod reg r/m] 8 [00sw] [mod 001 r/m] ### 0 [110w] ###	0 - - - x x u x 0	1 1 1 1 1	1 1 1 1 1	b	h
OUT <i>Output to Port</i> Fixed Port Variable Port	E [011w] # E [111w]	- - - - - - - -	14 14	14/28 14/28		m
OUTS <i>Output String</i>	6 [111w]	- - - - - - - -	14	14/28	b	h,m
POP <i>Pop Value off Stack</i> Register/Memory Register (short form) Segment Register (ES, SS, DS) Segment Register (FS, GS)	8F [mod 000 r/m] 5 [1 reg] [000 sreg2 111] 0F [10 sreg3 001]	- - - - - - - -	1 1 1 1	1 1 3 3	b	h,i,j
POPA <i>Pop All General Registers</i>	61	- - - - - - - -	6	6	b	h
POPF <i>Pop Stack into FLAGS</i>	9D	x x x x x x x x x	9	9	b	h,n

= immediate 8-bit data
= immediate 16-bit data
= full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
PREFIX BYTES Assert Hardware LOCK Prefix Address Size Prefix Operand Size Prefix Segment Override Prefix CS DS ES FS GS SS	F0 67 66 2E 3E 26 64 65 36	- - - - - - - -				m
PUSH <i>Push Value onto Stack</i> Register/Memory Register (short form) Segment Register (ES, CS, SS, DS) Segment Register (FS, GS) Immediate	FF [mod 110 r/m] 5 [0 reg] [000 sreg2 110] 0F [10 sreg3 000] 6 [10s0] ###	- - - - - - - -	1 1 1 1 1	1 1 1 1 1	b	h
PUSHA <i>Push All General Registers</i>	60	- - - - - - - -	6	6	b	h
PUSHF <i>Push FLAGS Register</i>	9C	- - - - - - - -	2	2	b	h
RCL <i>Rotate Through Carry Left</i> Register/Memory by 1 Register/Memory by CL Register/Memory by Immediate	D [000w] [mod 010 r/m] D [001w] [mod 010 r/m] C [000w] [mod 010 r/m] #	x - - - - - - x u - - - - - - x u - - - - - - x	3 8 8	3 8 8	b	h
RCR <i>Rotate Through Carry Right</i> Register/Memory by 1 Register/Memory by CL Register/Memory by Immediate	D [000w] [mod 011 r/m] D [001w] [mod 011 r/m] C [000w] [mod 011 r/m] #	x - - - - - - x u - - - - - - x u - - - - - - x	4 9 9	4 9 9	b	h
RDMSR <i>Read Model Specific Register</i>	0F 32	- - - - - - - -				
RDPMSR <i>Read Performance-Monitoring Counters</i>	0F 33	- - - - - - - -				
RDSHR <i>Read SMM Header Pointer Register</i>	0F 36	- - - - - - - -				
RDTSR <i>Read Time Stamp Counter</i>	0F 31	- - - - - - - -				
REP INS <i>Input String</i>	F3 6[110w]	- - - - - - - -	12+5n	12+5n\ 28+5n	b	h,m
REP LODS <i>Load String</i>	F3 A[110w]	- - - - - - - -	10+n	10+n	b	h
REP MOVS <i>Move String</i>	F3 A[010w]	- - - - - - - -	9+n	9+n	b	h
REP OUTS <i>Output String</i>	F3 6[111w]	- - - - - - - -	12+5n	12+5n\ 28+5n	b	h,m
REP STOS <i>Store String</i>	F3 A[101w]	- - - - - - - -	10+n	10+n	b	h
REPE CMPS <i>Compare String</i> (Find non-match)	F3 A[011w]	x - - - x x x x x	10+2n	10+2n	b	h

= immediate 8-bit data
= immediate 16-bit data
= full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/Cache Hit	Reg/Cache Hit	Real Mode	Protected Mode
REPE SCAS Scan String (Find non-AL/AX/EAX)	F3 A[111w]	x - - - x x x x x	10+2n	10+2n	b	h
REPNE CMPS Compare String (Find match)	F2 A[011w]	x - - - x x x x x	10+2n	10+2n	b	h
REPNE SCAS Scan String (Find AL/AX/EAX)	F2 A[111w]	x - - - x x x x x	10+2n	10+2n	b	h
RET Return from Subroutine Within Segment Within Segment Adding Immediate to SP Intersegment Intersegment Adding Immediate to SP Protected Mode: Different Privilege Level Intersegment Intersegment Adding Immediate to SP	C3 C2 ## CB CA ##	- - - - - - - - -	3 4 4 4	3 4 7 7	b	g,h,j,k,r
ROL Rotate Left Register/Memory by 1 Register/Memory by CL Register/Memory by Immediate	D[000w] [mod 000 r/m] D[001w] [mod 000 r/m] C[000w] [mod 000 r/m] #	x - - - - - - - x u - - - - - - - x u - - - - - - - x	1 2 1	1 2 1	b	h
ROR Rotate Right Register/Memory by 1 Register/Memory by CL Register/Memory by Immediate	D[000w] [mod 001 r/m] D[001w] [mod 001 r/m] C[000w] [mod 001 r/m] #	x - - - - - - - x u - - - - - - - x u - - - - - - - x	1 2 1	1 2 1	b	h
RSDC Restore Segment Register and Descriptor	0F 79 [mod sreg3 r/m]	- - - - - - - - -	6	6	s	s
RSLDT Restore LDTR and Descriptor	0F 7B [mod 000 r/m]	- - - - - - - - -	6	6	s	s
RSM Resume from SMM Mode	0F AA	x x x x x x x x x	40	40	s	s
RSTS Restore TSR and Descriptor	0F 7D [mod 000 r/m]	- - - - - - - - -	6	6	s	s
SAHF Store AH in FLAGS	9E	- - - - x x x x x	1	1		
SAL Shift Left Arithmetic Register/Memory by 1 Register/Memory by CL Register/Memory by Immediate	D[000w] [mod 100 r/m] D[001w] [mod 100 r/m] C[000w] [mod 100 r/m] #	x - - - x x u x x u - - - x x u x x u - - - x x u x x	1 2 1	1 2 1	b	h
SAR Shift Right Arithmetic Register/Memory by 1 Register/Memory by CL Register/Memory by Immediate	D[000w] [mod 111 r/m] D[001w] [mod 111 r/m] C[000w] [mod 111 r/m] #	x - - - x x u x x u - - - x x u x x u - - - x x u x x	1 2 1	1 2 1	b	h
SBB Integer Subtract with Borrow Register to Register Register to Memory Memory to Register Immediate to Register/Memory Immediate to Accumulator (short form)	1[10dw] [11 reg r/m] 1[100w] [mod reg r/m] 1[101w] [mod reg r/m] 8[00sw] [mod 011 r/m] ### 1[110w] ###	x - - - x x x x x	1 1 1 1 1	1 1 1 1 1	b	h

= immediate 8-bit data
= immediate 16-bit data
= full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
SCAS Scan String	A [111w]	x - - - x x x x x	2	2	b	h
SETB/SETNAE/SETC Set Byte on Below/Not Above or Equal/Carry To Register/Memory	0F 92 [mod 000 r/m]	- - - - - - - -	1	1		h
SETBE/SETNA Set Byte on Below or Equal/Not Above To Register/Memory	0F 96 [mod 000 r/m]	- - - - - - - -	1	1		h
SETE/SETZ Set Byte on Equal/Zero To Register/Memory	0F 94 [mod 000 r/m]	- - - - - - - -	1	1		h
SETL/SETNGE Set Byte on Less/Not Greater or Equal To Register/Memory	0F 9C [mod 000 r/m]	- - - - - - - -	1	1		h
SETLE/SETNG Set Byte on Less or Equal/Not Greater To Register/Memory	0F 9E [mod 000 r/m]	- - - - - - - -	1	1		h
SETNB/SETAE/SETNC Set Byte on Not Below/Above or Equal/Not Carry To Register/Memory	0F 93 [mod 000 r/m]	- - - - - - - -	1	1		h
SETNBE/SETA Set Byte on Not Below or Equal/Above To Register/Memory	0F 97 [mod 000 r/m]	- - - - - - - -	1	1		h
SETNE/SETNZ Set Byte on Not Equal/Not Zero To Register/Memory	0F 95 [mod 000 r/m]	- - - - - - - -	1	1		h
SETNL/SETGE Set Byte on Not Less/Greater or Equal To Register/Memory	0F 9D [mod 000 r/m]	- - - - - - - -	1	1		h
SETNLE/SETG Set Byte on Not Less or Equal/Greater To Register/Memory	0F 9F [mod 000 r/m]	- - - - - - - -	1	1		h
SETNO Set Byte on Not Overflow To Register/Memory	0F 91 [mod 000 r/m]	- - - - - - - -	1	1		h
SETNP/SETPO Set Byte on Not Parity/Parity Odd To Register/Memory	0F 9B [mod 000 r/m]	- - - - - - - -	1	1		h
SETNS Set Byte on Not Sign To Register/Memory	0F 99 [mod 000 r/m]	- - - - - - - -	1	1		h
SETO Set Byte on Overflow To Register/Memory	0F 90 [mod 000 r/m]	- - - - - - - -	1	1		h

= immediate 8-bit data
= immediate 16-bit data
= full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/Cache Hit	Reg/Cache Hit	Real Mode	Protected Mode
SETP/SETPE Set Byte on Parity/Parity Even To Register/Memory	0F 9A [mod 000 r/m]	- - - - - - - -	1	1		h
SETS Set Byte on Sign To Register/Memory	0F 98 [mod 000 r/m]	- - - - - - - -	1	1		h
SGDT Store GDT Register To Register/Memory	0F 01 [mod 000 r/m]	- - - - - - - -	4	4	b,c	h
SHL Shift Left Logical Register/Memory by 1 Register/Memory by CL Register/Memory by Immediate	D [000w] [mod 100 r/m]	x - - - x x u x x	1	1	b	h
	D [001w] [mod 100 r/m]	u - - - x x u x x	2	2		
	C [000w] [mod 100 r/m] #	u - - - x x u x x	1	1		
SHLD Shift Left Double Register/Memory by Immediate Register/Memory by CL	0F A4 [mod reg r/m] #	u - - - x x u x x	4	4	b	h
	0F A5 [mod reg r/m]		5	5		
SHR Shift Right Logical Register/Memory by 1 Register/Memory by CL Register/Memory by Immediate	D [000w] [mod 101 r/m]	x - - - x x u x x	1	1	b	h
	D [001w] [mod 101 r/m]	u - - - x x u x x	2	2		
	C [000w] [mod 101 r/m] #	u - - - x x u x x	1	1		
SHRD Shift Right Double Register/Memory by Immediate Register/Memory by CL	0F AC [mod reg r/m] #	u - - - x x u x x	4	4	b	h
	0F AD [mod reg r/m]		5	5		
SIDT Store IDT Register To Register/Memory	0F 01 [mod 001 r/m]	- - - - - - - -	4	4	b,c	h
SLDT Store LDT Register To Register/Memory	0F 00 [mod 000 r/m]	- - - - - - - -		1	a	h
SMINT Software SMM Entry	0F 38	- - - - - - - -	55	55	s	s
SMSW Store Machine Status Word	0F 01 [mod 100 r/m]	- - - - - - - -	6	6	b,c	h
STC Set Carry Flag	F9	- - - - - - - 1	1	1		
STD Set Direction Flag	FD	- 1 - - - - - -	7	7		
STI Set Interrupt Flag	FB	- - 1 - - - - -	7	7		m
STOS Store String	A [101w]	- - - - - - - -	2	2	b	h
STR Store Task Register To Register/Memory	0F 00 [mod 001 r/m]	- - - - - - - -		4	a	h

= immediate 8-bit data
= immediate 16-bit data
= full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined

Table 6-21. 6x86MX CPU Instruction Set Clock Count Summary (Continued)

INSTRUCTION	OPCODE	FLAGS	REAL MODE CLOCK COUNT	PROTECTED MODE CLOCK COUNT	NOTES	
		OF DF IF TF SF ZF AF PF CF	Reg/ Cache Hit	Reg/ Cache Hit	Real Mode	Protected Mode
SUB Integer Subtract Register to Register Register to Memory Memory to Register Immediate to Register/Memory Immediate to Accumulator (short form)	2 [10dw] [11 reg r/m] 2 [100w] [mod reg r/m] 2 [101w] [mod reg r/m] 8 [00sw] [mod 101 r/m] ### 2 [110w] ###	x - - - x x x x x	1 1 1 1 1	1 1 1 1 1	b	h
SVDC Save Segment Register and Descriptor	0F 78 [mod sreg3 r/m]	- - - - - - - - -	12	12	s	s
SVLDT Save LDTR and Descriptor	0F 7A [mod 000 r/m]	- - - - - - - - -	12	12	s	s
SVTS Save TSR and Descriptor	0F 7C [mod 000 r/m]	- - - - - - - - -	14	14	s	s
TEST Test Bits Register/Memory and Register Immediate Data and Register/Memory Immediate Data and Accumulator	8 [010w] [mod reg r/m] F [011w] [mod 000 r/m] ### A [100w] ###	0 - - - x x u x 0	1 1 1	1 1 1	b	h
VERR Verify Read Access To Register/Memory	0F 00 [mod 100 r/m]	- - - - - x - - -		7	a	g,h,j,p
VERW Verify Write Access To Register/Memory	0F 00 [mod 101 r/m]	- - - - - x - - -		7	a	g,h,j,p
WAIT Wait Until FPU Not Busy	9B	- - - - - - - - -	5	5		
WBINVD Write-Back and Invalidate Cache	0F 09	- - - - - - - - -	15	15	t	t
WRMSR Write to Model Specific Register	0F 30	- - - - - - - - -				
WRSHR Write SMM Header Pointer Register	0F 37	- - - - - - - - -				
XADD Exchange and Add Register1, Register2 Memory, Register	0F C[000w] [11 reg2 reg1] 0F C[000w] [mod reg r/m]	x - - - x x x x x	2 2	2 2		
XCHG Exchange Register/Memory with Register Register with Accumulator	8[011w] [mod reg r/m] 9[0 reg]	- - - - - - - - -	2 2	2 2	b,f	f,h
XLAT Translate Byte	D7	- - - - - - - - -	4	4		h
XOR Boolean Exclusive OR Register to Register Register to Memory Memory to Register Immediate to Register/Memory Immediate to Accumulator (short form)	3 [00dw] [11 reg r/m] 3 [000w] [mod reg r/m] 3 [001w] [mod reg r/m] 8 [00sw] [mod 110 r/m] ### 3 [010w] ###	0 - - - x x u x 0	1 1 1 1 1	1 1 1 1 1	b	h

= immediate 8-bit data
= immediate 16-bit data
= full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined

Instruction Notes for Instruction Set Summary**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 op-code).
- 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 segment 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.
- d. -

Notes e through g apply to Real Address Mode and Protected Virtual Address Mode:

- 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 will occur if the memory operand in CS, DS, ES, FS, or GS cannot be used due to either a segment limit violation or an access rights violation. If a stack limit is violated, an exception 12 occurs.
- i. For segment load operations, the CPL, RPL, and DPL must agree with the privilege rules to avoid an exception 13 fault. 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 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, 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 (CRO) cannot be reset by this instruction. Use MOV into CRO if desiring to reset the PE bit.
- p. Any violation of privilege rules as apply 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 will occur before the ESC instruction is executed. An exception 12 fault 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 will occur.

Note s applies to Cyrix specific SMM instructions:

- s. All memory accesses to SMM space are non-cacheable. An invalid opcode exception 6 occurs unless SMI is enabled and ARR3 size > 0, and CPL = 0 and [SMAC is set or if in an SMI handler].

Note t applies to cache invalidation instructions with the cache operating in write-back mode:

- t. The total clock count is the clock count shown plus the number of clocks required to write all "modified" cache lines to external memory.

6.5 FPU Instruction Clock Counts

The CPU is functionally divided into the FPU unit, and the integer unit. The FPU has been extended to processes MMX instructions as well as floating point instructions in parallel with the integer unit.

For example, when the integer unit detects a floating point instruction the instruction passes to the FPU for execution. The integer unit continues to execute instructions while the FPU executes the floating point instruction.

If another FPU instruction is encountered, the second FPU instruction is placed in the FPU queue. Up to four FPU instructions can be queued. In the event of an FPU exception, while other FPU instructions are queued, the state of the CPU is saved to ensure recovery.

6.5.1 FPU Clock Count Table

The clock counts for the FPU instructions are listed in Table 6-23 (Page 6-31). The abbreviations used in this table are listed in Table 6-22.

Table 6-22. FPU Clock Count Table Abbreviations

ABBREVIATION	MEANING
n	Stack register number
TOS	Top of stack register pointed to by SSS in the status register.
ST(1)	FPU register next to TOS
ST(n)	A specific FPU register, relative to TOS
M.WI	16-bit integer operand from memory
M.SI	32-bit integer operand from memory
M.LI	64-bit integer operand from memory
M.SR	32-bit real operand from memory
M.DR	64-bit real operand from memory
M.XR	80-bit real operand from memory
M.BCD	18-digit BCD integer operand from memory
CC	FPU condition code
Env Regs	Status, Mode Control and Tag Registers, Instruction Pointer and Operand Pointer

Table 6-23. 6x86MX FPU Instruction Set Summary

FPU INSTRUCTION	OP CODE	OPERATION	CLOCK COUNT	NOTES
F2XMI Function Evaluation 2^{X-1}	D9 F0	$TOS \leftarrow 2^{TOS-1}$	92 - 108	See Note 2
FABS Floating Absolute Value	D9 E1	$TOS \leftarrow TOS $	2	
FADD Floating Point Add				
Top of Stack	DC [1100 0 n]	$ST(n) \leftarrow ST(n) + TOS$	4 - 7	
80-bit Register	D8 [1100 0 n]	$TOS \leftarrow TOS + ST(n)$	4 - 7	
64-bit Real	DC [mod 000 r/m]	$TOS \leftarrow TOS + M.DR$	4 - 7	
32-bit Real	D8 [mod 000 r/m]	$TOS \leftarrow TOS + M.SR$	4 - 7	
FADDP Floating Point Add, Pop	DE [1100 0 n]	$ST(n) \leftarrow ST(n) + TOS$; then pop TOS	4 - 7	
FIADD Floating Point Integer Add				
32-bit integer	DA [mod 000 r/m]	$TOS \leftarrow TOS + M.SI$	8 - 12	
16-bit integer	DE [mod 000 r/m]	$TOS \leftarrow TOS + M.WI$	8 - 12	
FCHS Floating Change Sign	D9 E0	$TOS \leftarrow -TOS$	2	
FCLEX Clear Exceptions	(9B)DB E2	Wait then Clear Exceptions	5	
FNCLEX Clear Exceptions	DB E2	Clear Exceptions	3	
FCOM Floating Point Compare				
80-bit Register	D8 [1101 0 n]	CC set by $TOS - ST(n)$	4	
64-bit Real	DC [mod 010 r/m]	CC set by $TOS - M.DR$	4	
32-bit Real	D8 [mod 010 r/m]	CC set by $TOS - M.SR$	4	
FCOMP Floating Point Compare, Pop				
80-bit Register	D8 [1101 1 n]	CC set by $TOS - ST(n)$; then pop TOS	4	
64-bit Real	DC [mod 011 r/m]	CC set by $TOS - M.DR$; then pop TOS	4	
32-bit Real	D8 [mod 011 r/m]	CC set by $TOS - M.SR$; then pop TOS	4	
FCOMPP Floating Point Compare, Pop Two Stack Elements	DE D9	CC set by $TOS - ST(1)$; then pop TOS and $ST(1)$	4	
FICOM Floating Point Compare				
32-bit integer	DA [mod 010 r/m]	CC set by $TOS - M.WI$	9 - 10	
16-bit integer	DE [mod 010 r/m]	CC set by $TOS - M.SI$	9 - 10	
FICOMP Floating Point Compare				
32-bit integer	DA [mod 011 r/m]	CC set by $TOS - M.WI$; then pop TOS	9 - 10	
16-bit integer	DE [mod 011 r/m]	CC set by $TOS - M.SI$; then pop TOS	9 - 10	
FCOMI Floating Point Compare Real and Set EFLAGS				
80-bit Register	DB [1111 0 n]	EFLAG set by $TOS - ST(n)$	4	
FCOMIP Floating Point Compare Real and Set EFLAGS, Pop				
80-bit Register	DF [1111 0 n]	EFLAG set by $TOS - ST(n)$; then pop TOS	4	
FUCOMI Floating Point Unordered Compare Real and Set EFLAGS				
80-bit integer	DB [1110 1 n]	EFLAG set by $TOS - ST(n)$	9 - 10	
FUCOMIP Floating Point Unordered Compare Real and Set EFLAGS				
80-bit integer	DF [1110 1 n]	EFLAG set by $TOS - ST(n)$; then pop TOS	9 - 10	
FCMOVB Floating Point Conditional Move if Below	DA [1100 0 n]	If $(CF=1) ST(0) \leftarrow ST(n)$	4	
FCMOVE Floating Point Conditional Move if Equal	DA [1100 1 n]	If $(ZF=1) ST(0) \leftarrow ST(n)$	4	

Table 6-23. 6x86MX FPU Instruction Set Summary (Continued)

FPU INSTRUCTION	OP CODE	OPERATION	CLOCK COUNT	NOTES
FCMOVBE Floating Point Conditional Move if Below or Equal	DA [1101 0 n]	If (CF=1 or ZF=1) ST(0) ←—ST(n)	4	
FCMOVU Floating Point Conditional Move if Unordered	DA [1101 1 n]	If (PF=1) ST(0) ←—ST(n)	4	
FCMOVNB Floating Point Conditional Move if Not Below	DB [1100 0 n]	If (CF=0) ST(0) ←—ST(n)	4	
FCMOVNE Floating Point Conditional Move if Not Equal	DB [1100 1 n]	If (ZF=0) ST(0) ←—ST(n)	4	
FCMOVNBE Floating Point Conditional Move if Not Below or Equal	DB [1101 0 n]	If (CF=0 and ZF=0) ST(0) ←—ST(n)	4	
FCMOVNU Floating Point Conditional Move if Not Unordered	DB [1101 1 n]	If (DF=0) ST(0) ←—ST(n)	4	
FCOS Function Evaluation: Cos(x)	D9 FF	TOS ← COS(TOS)	92 - 141	See Note 1
FDECSTP Decrement Stack Pointer	D9 F6	Decrement top of stack pointer	4	
FDIV Floating Point Divide Top of Stack 80-bit Register 64-bit Real 32-bit Real	DC [1111 1 n] D8 [1111 0 n] DC [mod 110 r/m] D8 [mod 110 r/m]	ST(n) ← ST(n) / TOS TOS ← TOS / ST(n) TOS ← TOS / M.DR TOS ← TOS / M.SR	24 - 34 24 - 34 24 - 34 24 - 34	
FDIVP Floating Point Divide, Pop	DE [1111 1 n]	ST(n) ← ST(n) / TOS; then pop TOS	24 - 34	
FDIVR Floating Point Divide Reversed Top of Stack 80-bit Register 64-bit Real 32-bit Real	DC [1111 0 n] D8 [1111 1 n] DC [mod 111 r/m] D8 [mod 111 r/m]	TOS ← ST(n) / TOS ST(n) ← TOS / ST(n) TOS ← M.DR / TOS TOS ← M.SR / TOS	24 - 34 24 - 34 24 - 34 24 - 34	
FDIVRP Floating Point Divide Reversed, Pop	DE [1111 0 n]	ST(n) ← TOS / ST(n); then pop TOS	24 - 34	
FIDIV Floating Point Integer Divide 32-bit Integer 16-bit Integer	DA [mod 110 r/m] DE [mod 110 r/m]	TOS ← TOS / M.SI TOS ← TOS / M.WI	34 - 38 33 - 38	
FIDIVR Floating Point Integer Divide Reversed 32-bit Integer 16-bit Integer	DA [mod 111 r/m] DE [mod 111 r/m]	TOS ← M.SI / TOS TOS ← M.WI / TOS	34 - 38 33 - 38	
FFREE Free Floating Point Register	DD [1100 0 n]	TAG(n) ← Empty	3	
FINCSTP Increment Stack Pointer	D9 F7	Increment top of stack pointer	2	
FINIT Initialize FPU	(9B)DB E3	Wait then initialize	8	
FNINIT Initialize FPU	DB E3	Initialize	6	

Table 6-23. 6x86MX FPU Instruction Set Summary (Continued)

FPU INSTRUCTION	OP CODE	OPERATION	CLOCK COUNT	NOTES
FLD Load Data to FPU Reg. Top of Stack	D9 [1100 0 n]	Push ST(n) onto stack	2	
64-bit Real	DD [mod 000 r/m]	Push M.DR onto stack	2	
32-bit Real	D9 [mod 000 r/m]	Push M.SR onto stack	2	
FBLD Load Packed BCD Data to FPU Reg.	DF [mod 100 r/m]	Push M.BCD onto stack	41 - 45	
FILD Load Integer Data to FPU Reg. 64-bit Integer	DF [mod 101 r/m]	Push M.LI onto stack	4 - 8	
32-bit Integer	DB [mod 000 r/m]	Push M.SI onto stack	4 - 6	
16-bit Integer	DF [mod 000 r/m]	Push M.WI onto stack	3 - 6	
FLDI Load Floating Const.= 1.0	D9 E8	Push 1.0 onto stack	4	
FLDCW Load FPU Mode Control Register	D9 [mod 101 r/m]	Ctl Word ← Memory	4	
FLDENV Load FPU Environment	D9 [mod 100 r/m]	Env Regs ← Memory	30	
FLDL2E Load Floating Const.= $\log_2(e)$	D9 EA	Push $\log_2(e)$ onto stack	4	
FLDL2T Load Floating Const.= $\log_2(10)$	D9 E9	Push $\log_2(10)$ onto stack	4	
FLDLG2 Load Floating Const.= $\log_{10}(2)$	D9 EC	Push $\log_{10}(2)$ onto stack	4	
FLDLN2 Load Floating Const.= $\ln(2)$	D9 ED	Push $\log_e(2)$ onto stack	4	
FLDPI Load Floating Const.= π	D9 EB	Push π onto stack	4	
FLDZ Load Floating Const.= 0.0	D9 EE	Push 0.0 onto stack	4	
FMUL Floating Point Multiply Top of Stack	DC [1100 1 n]	ST(n) ← ST(n) × TOS	4 - 6	
80-bit Register	D8 [1100 1 n]	TOS ← TOS × ST(n)	4 - 6	
64-bit Real	DC [mod 001 r/m]	TOS ← TOS × M.DR	4 - 6	
32-bit Real	D8 [mod 001 r/m]	TOS ← TOS × M.SR	4 - 5	
FMULP Floating Point Multiply & Pop	DE [1100 1 n]	ST(n) ← ST(n) × TOS; then pop TOS	4 - 6	
FIMUL Floating Point Integer Multiply 32-bit Integer	DA [mod 001 r/m]	TOS ← TOS × M.SI	9 - 11	
16-bit Integer	DE [mod 001 r/m]	TOS ← TOS × M.WI	8 - 10	
FNOP No Operation	D9 D0	No Operation	2	
FPATAN Function Eval: $\tan^{-1}(y/x)$	D9 F3	ST(1) ← ATAN[ST(1) / TOS]; then pop TOS	97 - 161	See Note 3
FPREM Floating Point Remainder	D9 F8	TOS ← Rem[TOS / ST(1)]	82 - 91	
FPREMI Floating Point Remainder IEEE	D9 F5	TOS ← Rem[TOS / ST(1)]	82 - 91	
FPTAN Function Eval: $\tan(x)$	D9 F2	TOS ← TAN(TOS); then push 1.0 onto stack	117 - 129	See Note 1
FRNDINT Round to Integer	D9 FC	TOS ← Round(TOS)	10 - 20	
FRSTOR Load FPU Environment and Reg.	DD [mod 100 r/m]	Restore state.	56 - 72	
FSAVE Save FPU Environment and Reg	(9B)DD[mod 110 r/m]	Wait then save state.	57 - 67	
FNSAVE Save FPU Environment and Reg	DD [mod 110 r/m]	Save state.	55 - 65	
FSCALE Floating Multiply by 2^n	D9 FD	TOS ← TOS × $2^{ST(1)}$	7 - 14	
FSIN Function Evaluation: $\sin(x)$	D9 FE	TOS ← SIN(TOS)	76 - 140	See Note 1
FSINCOS Function Eval.: $\sin(x)$ & $\cos(x)$	D9 FB	temp ← TOS; TOS ← SIN(temp); then push COS(temp) onto stack	145 - 161	See Note 1
FSQRT Floating Point Square Root	D9 FA	TOS ← Square Root of TOS	59 - 60	

Table 6-23. 6x86MX FPU Instruction Set Summary (Continued)

FPU INSTRUCTION	OP CODE	OPERATION	CLOCK COUNT	NOTES
FST Store FPU Register				
Top of Stack	DD [1101 0 n]	ST(n) ← TOS	2	
80-bit Real	DB [mod 111 r/m]	M.XR ← TOS	2	
64-bit Real	DD [mod 010 r/m]	M.DR ← TOS	2	
32-bit Real	D9 [mod 010 r/m]	M.SR ← TOS	2	
FSTP Store FPU Register, Pop				
Top of Stack	DB [1101 1 n]	ST(n) ← TOS; then pop TOS	2	
80-bit Real	DB [mod 111 r/m]	M.XR ← TOS; then pop TOS	2	
64-bit Real	DD [mod 011 r/m]	M.DR ← TOS; then pop TOS	2	
32-bit Real	D9 [mod 011 r/m]	M.SR ← TOS; then pop TOS	2	
FBSTP Store BCD Data, Pop	DF [mod 110 r/m]	M.BCD ← TOS; then pop TOS	57 - 63	
FIST Store Integer FPU Register				
32-bit Integer	DB [mod 010 r/m]	M.SI ← TOS	8 - 13	
16-bit Integer	DF [mod 010 r/m]	M.WI ← TOS	7 - 10	
FISTP Store Integer FPU Register, Pop				
64-bit Integer	DF [mod 111 r/m]	M.LI ← TOS; then pop TOS	10 - 13	
32-bit Integer	DB [mod 011 r/m]	M.SI ← TOS; then pop TOS	8 - 13	
16-bit Integer	DF [mod 011 r/m]	M.WI ← TOS; then pop TOS	7 - 10	
FSTCW Store FPU Mode Control Register	(9B)D9[mod 111 r/m]	Wait Memory ← Control Mode Register	5	
FNSTCW Store FPU Mode Control Register	D9 [mod 111 r/m]	Memory ← Control Mode Register	3	
FSTENV Store FPU Environment	(9B)D9[mod 110 r/m]	Wait Memory ← Env. Registers	14 - 24	
FNSTENV Store FPU Environment	D9 [mod 110 r/m]	Memory ← Env. Registers	12 - 22	
FSTSW Store FPU Status Register	(9B)DD[mod 111 r/m]	Wait Memory ← Status Register	6	
FNSTSW Store FPU Status Register	DD [mod 111 r/m]	Memory ← Status Register	4	
FSTSW AX Store FPU Status Register to AX	(9B)DF E0	Wait AX ← Status Register	4	
FNSTSW AX Store FPU Status Register to AX	DF E0	AX ← Status Register	2	
FSUB Floating Point Subtract				
Top of Stack	DC [1110 1 n]	ST(n) ← ST(n) - TOS	4 - 7	
80-bit Register	D8 [1110 0 n]	TOS ← TOS - ST(n)	4 - 7	
64-bit Real	DC [mod 100 r/m]	TOS ← TOS - M.DR	4 - 7	
32-bit Real	D8 [mod 100 r/m]	TOS ← TOS - M.SR	4 - 7	
FSUBP Floating Point Subtract, Pop	DE [1110 1 n]	ST(n) ← ST(n) - TOS; then pop TOS	4 - 7	

Table 6-23. 6x86MX FPU Instruction Set Summary (Continued)

FPU INSTRUCTION	OP CODE	OPERATION	CLOCK COUNT	NOTES
FSUBR <i>Floating Point Subtract Reverse</i> Top of Stack	DC [1110 0 n]	$TOS \leftarrow ST(n) - TOS$	4 - 7	
80-bit Register	D8 [1110 1 n]	$ST(n) \leftarrow TOS - ST(n)$	4 - 7	
64-bit Real	DC [mod 101 r/m]	$TOS \leftarrow M.DR - TOS$	4 - 7	
32-bit Real	D8 [mod 101 r/m]	$TOS \leftarrow M.SR - TOS$	4 - 7	
FSUBRP <i>Floating Point Subtract Reverse, Pop</i>	DE [1110 0 n]	$ST(n) \leftarrow TOS - ST(n)$; then pop TOS	4 - 7	
FISUB <i>Floating Point Integer Subtract</i> 32-bit Integer	DA [mod 100 r/m]	$TOS \leftarrow TOS - M.SI$	14 - 29	
16-bit Integer	DE [mod 100 r/m]	$TOS \leftarrow TOS - M.WI$	14 - 27	
FISUBR <i>Floating Point Integer Subtract Reverse</i> 32-bit Integer Reversed	DA [mod 101 r/m]	$TOS \leftarrow M.SI - TOS$	14 - 29	
16-bit Integer Reversed	DE [mod 101 r/m]	$TOS \leftarrow M.WI - TOS$	14 - 27	
FTST <i>Test Top of Stack</i>	D9 E4	CC set by $TOS - 0.0$	4	
FUCOM <i>Unordered Compare</i>	DD [1110 0 n]	CC set by $TOS - ST(n)$	4	
FUCOMP <i>Unordered Compare, Pop</i>	DD [1110 1 n]	CC set by $TOS - ST(n)$; then pop TOS	4	
FUCOMPP <i>Unordered Compare, Pop two elements</i>	DA E9	CC set by $TOS - ST(1)$; then pop TOS and $ST(1)$	4	
FWAIT <i>Wait</i>	9B	Wait for FPU not busy	2	
FXAM <i>Report Class of Operand</i>	D9 E5	$CC \leftarrow$ Class of TOS	4	
FXCH <i>Exchange Register with TOS</i>	D9 [1100 1 n]	$TOS \leftrightarrow ST(n)$ Exchange	2	
EXTRACT <i>Extract Exponent</i>	D9 F4	temp \leftarrow TOS; $TOS \leftarrow$ exponent (temp); then push significant (temp) onto stack	11 - 16	
FLY2X <i>Function Eval. $y \times \text{Log}_2(x)$</i>	D9 F1	$ST(1) \leftarrow ST(1) \times \text{Log}_2(TOS)$; then pop TOS	145 - 154	
FLY2XP1 <i>Function Eval. $y \times \text{Log}_2(x+1)$</i>	D9 F9	$ST(1) \leftarrow ST(1) \times \text{Log}_2(1+TOS)$; then pop TOS	131 - 133	See Note 4

FPU Instruction Summary Notes

All references to TOS and ST(n) refer to stack layout prior to execution.

Values popped off the stack are discarded.

A pop from the stack increments the top of stack pointer.

A push to the stack decrements the top of stack pointer.

Note 1:

For FCOS, FSIN, FSINCOS and FPTAN, time shown is for absolute value of TOS < $3\pi/4$. Add 90 clock counts for argument reduction if outside this range.

For FCOS, clock count is 141 if TOS < $\pi/4$ and clock count is 92 if $\pi/4 < \text{TOS} < \pi/2$.

For FSIN, clock count is 81 to 82 if absolute value of TOS < $\pi/4$.

Note 2:

For F2XM1, clock count is 92 if absolute value of TOS < 0.5.

Note 3:

For FPATAN, clock count is 97 if ST(1)/TOS < $\pi/32$.

Note 4:

For FYL2XP1, clock count is 170 if TOS is out of range and regular FYL2X is called.

Note 5:

The following opcodes are reserved by Cyrix:

D9D7, D9E2, D9E7, DDFC, DED8, DEDA, DEDC, DEDD, DEDE, DFFC.

If a reserved opcode is executed, and unpredictable results may occur (exceptions are not generated).

6.6 6x86MX Processor MMX Instruction Clock Counts

The CPU is functionally divided into the FPU unit, and the integer unit. The FPU has been extended to process both MMX instructions and floating point instructions in parallel with the integer unit.

For example, when the integer unit detects a MMX instruction, the instruction passes to the FPU unit for execution. The integer unit con-

tinues to execute instructions while the FPU unit executes the MMX instruction. If another MMX instruction is encountered, the second MMX instruction is placed in the MMX queue. Up to four MMX instructions can be queued.

6.6.1 MMX Clock Count Table

The clock counts for the MMX instructions are listed in Table 6-25 (Page 38). The abbreviations used in this table are listed in Table 6-24.

Table 6-24. MMX Clock Count Table Abbreviations

ABBREVIATION	MEANING
<---	Result written
[11 mm reg]	Binary or binary groups of digits
mm	One of eight 64-bit MMX registers
reg	A general purpose register
<--sat--	If required, the resultant data is saturated to remain in the associated data range
<--move--	Source data is moved to result location
[byte]	Eight 8-bit bytes are processed in parallel
[word]	Four 16-bit word are processed in parallel
[dword]	Two 32-bit double words are processed in parallel
[qword]	One 64-bit quad word is processed
[sign xxx]	The byte, word, double word or quad word most significant bit is a sign bit
mm1, mm2	MMX register 1, MMX register 2
mod r/m	Mod and r/m byte encoding (page 6-6 of this manual)
pack	Source data is truncated or saturated to next smaller data size, then concatenated.
packdw	Pack two double words from source and two double words from destination into four words in destination register.
packwb	Pack four words from source and four words from destination into eight bytes in destination register.

Table 6-25. 6x86MX Processor MMX Instruction Set Clock Count Summary

MMX INSTRUCTIONS	OPCODE	OPERATION	CLOCK COUNT LATENCY/ THROUGHPUT
EMMS <i>Empty MMX State</i>	0F77	Tag Word <--- FFFFh (empties the floating point tag word)	1/1
MOVD <i>Move Doubleword</i> Register to MMX Register MMX Register to Register Memory to MMX Register MMX Register to Memory	0F6E [11 mm reg] 0F7E [11 mm reg] 0F6E [mod mm r/m] 0F7E [mod mm r/m]	MMX reg [qword] <--move, zero extend-- reg [dword] reg [qword] <--move-- MMX reg [low dword] MMX reg [qword] <--move, zero extend-- memory [dword] Memory [dword] <--move-- MMX reg [low dword]	1/1 5/1 1/1 1/1
MOVQ <i>Move Quadword</i> MMX Register 2 to MMX Register 1 MMX Register 1 to MMX Register 2 Memory to MMX Register MMX Register to Memory	0F6F [11 mm1 mm2] 0F7F [11 mm1 mm2] 0F6F [mod mm r/m] 0F7F [mod mm r/m]	MMX reg 1 [qword] <--move-- MMX reg 2 [qword] MMX reg 2 [qword] <--move-- MMX reg 1 [qword] MMX reg [qword] <--move-- memory [qword] Memory [qword] <--move-- MMX reg [qword]	1/1 1/1 1/1 1/1
PACKSSDW <i>Pack Dword with Signed Saturation</i> MMX Register 2 to MMX Register 1 Memory to MMX Register	0F6B [11 mm1 mm2] 0F6B [mod mm r/m]	MMX reg 1 [qword] <--packdw, signed sat-- MMX reg 2, MMX reg 1 MMX reg [qword] <--packdw, signed sat-- memory, MMX reg	1/1 1/1
PACKSWB <i>Pack Word with Signed Saturation</i> MMX Register 2 to MMX Register 1 Memory to MMX Register	0F63 [11 mm1 mm2] 0F63 [mod mm r/m]	MMX reg 1 [qword] <--packwb, signed sat-- MMX reg 2, MMX reg 1 MMX reg [qword] <--packwb, signed sat-- memory, MMX reg	1/1 1/1
PACKUSWB <i>Pack Word with Unsigned Saturation</i> MMX Register 2 to MMX Register 1 Memory to MMX Register	0F67 [11 mm1 mm2] 0F67 [mod mm r/m]	MMX reg 1 [qword] <--packwb, unsigned sat-- MMX reg 2, MMX reg 1 MMX reg [qword] <--packwb, unsigned sat-- memory, MMX reg	1/1 1/1
PADDB <i>Packed Add Byte with Wrap-Around</i> MMX Register 2 to MMX Register 1 Memory to MMX Register	0FFC [11 mm1 mm2] 0FFC [mod mm r/m]	MMX reg 1 [byte] <---- MMX reg 1 [byte] + MMX reg 2 [byte] MMX reg [byte] <---- memory [byte] + MMX reg [byte]	1/1 1/1
PADDD <i>Packed Add Dword with Wrap-Around</i> MMX Register 2 to MMX Register 1 Memory to MMX Register	0FFE [11 mm1 mm2] 0FFE [mod mm r/m]	MMX reg 1 [sign dword] <---- MMX reg 1 [sign dword] + MMX reg 2 [sign dword] MMX reg [sign dword] <---- memory [sign dword] + MMX reg [sign dword]	1/1 1/1
PADDSB <i>Packed Add Signed Byte with Saturation</i> MMX Register 2 to MMX Register 1 Memory to Register	0FEC [11 mm1 mm2] 0FEC [mod mm r/m]	MMX reg 1 [sign byte] <--sat-- MMX reg 1 [sign byte] + MMX reg 2 [sign byte] MMX reg [sign byte] <--sat-- memory [sign byte] + MMX reg [sign byte]	1/1 1/1
PADDSW <i>Packed Add Signed Word with Saturation</i> MMX Register 2 to MMX Register 1 Memory to Register	0FED [11 mm1 mm2] 0FED [mod mm r/m]	MMX reg 1 [sign word] <--sat-- MMX reg 1 [sign word] + MMX reg 2 [sign word] MMX reg [sign word] <--sat-- memory [sign word] + MMX reg [sign word]	1/1 1/1
PADDUSB <i>Add Unsigned Byte with Saturation</i> MMX Register 2 to MMX Register 1 Memory to Register	0FDC [11 mm1 mm2] 0FDC [mod mm r/m]	MMX reg 1 [byte] <--sat-- MMX reg 1 [byte] + MMX reg 2 [byte] MMX reg [byte] <--sat-- memory [byte] + MMX reg [byte]	1/1 1/1
PADDUSW <i>Add Unsigned Word with Saturation</i> MMX Register 2 to MMX Register 1 Memory to Register	0FDD [11 mm1 mm2] 0FDD [mod mm r/m]	MMX reg 1 [word] <--sat-- MMX reg 1 [word] + MMX reg 2 [word] MMX reg [word] <--sat-- memory [word] + MMX reg [word]	1/1 1/1

Table 6-25. 6x86MX Processor MMX Instruction Set Clock Count Summary (Continued)

MMX INSTRUCTIONS	OPCODE	OPERATION	CLOCK COUNT LATENCY/ THROUGHPUT
PADDW <i>Packed Add Word with Wrap-Around</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FFD [11 mm1 mm2] 0FFD [mod mm r/m]	MMX reg 1 [word] <---- MMX reg 1 [word] + MMX reg 2 [word] MMX reg [word] <---- memory [word] + MMX reg [word]	1/1 1/1
PAND <i>Bitwise Logical AND</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FDB [11 mm1 mm2] 0FDB [mod mm r/m]	MMX Reg 1 [qword] <--logic AND-- MMX Reg 1 [qword], MMX Reg 2 [qword] MMX Reg [qword] <--logic AND-- memory[qword], MMX Reg [qword]	1/1 1/1
PANDN <i>Bitwise Logical AND NOT</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FDF [11 mm1 mm2] 0FDF [mod mm r/m]	MMX Reg 1 [qword] <--logic AND -- NOT MMX Reg 1 [qword], MMX Reg 2 [qword] MMX Reg [qword] <--logic AND-- NOT MMX Reg [qword], Memory[qword]	1/1 1/1
PCMPEQB <i>Packed Byte Compare for Equality</i> MMX Register 2 with MMX Register1 Memory with MMX Register	0F74 [11 mm1 mm2] 0F74 [mod mm r/m]	MMX reg 1 [byte] <--FFh-- if MMX reg 1 [byte] = MMX reg 2 [byte] MMX reg 1 [byte]<--00h-- if MMX reg 1 [byte] NOT = MMX reg 2 [byte] MMX reg [byte] <--FFh-- if memory[byte] = MMX reg [byte] MMX reg [byte] <--00h-- if memory[byte] NOT = MMX reg [byte]	1/1 1/1
PCMPEQD <i>Packed Dword Compare for Equality</i> MMX Register 2 with MMX Register1 Memory with MMX Register	0F76 [11 mm1 mm2] 0F76 [mod mm r/m]	MMX reg 1 [dword] <--FFFF FFFFh-- if MMX reg 1 [dword] = MMX reg 2 [dword] MMX reg 1 [dword]<--0000 0000h--if MMX reg 1[dword] NOT = MMX reg 2 [dword] MMX reg [dword] <--FFFF FFFFh-- if memory[dword] = MMX reg [dword] MMX reg [dword] <--0000 0000h-- if memory[dword] NOT = MMX reg [dword]	1/1 1/1
PCMPEQW <i>Packed Word Compare for Equality</i> MMX Register 2 with MMX Register1 Memory with MMX Register	0F75 [11 mm1 mm2] 0F75 [mod mm r/m]	MMX reg 1 [word] <--FFFFh-- if MMX reg 1 [word] = MMX reg 2 [word] MMX reg 1 [word]<--0000h-- if MMX reg 1 [word] NOT = MMX reg 2 [word] MMX reg [word] <--FFFFh-- if memory[word] = MMX reg [word] MMX reg [word] <--0000h-- if memory[word] NOT = MMX reg [word]	1/1 1/1
PCMPGTB <i>Pack Compare Greater Than Byte</i> MMX Register 2 to MMX Register1 Memory with MMX Register	0F64 [11 mm1 mm2] 0F64 [mod mm r/m]	MMX reg 1 [byte] <--FFh-- if MMX reg 1 [byte] > MMX reg 2 [byte] MMX reg 1 [byte]<--00h-- if MMX reg 1 [byte] NOT > MMX reg 2 [byte] MMX reg [byte] <--FFh-- if memory[byte] > MMX reg [byte] MMX reg [byte] <--00h-- if memory[byte] NOT > MMX reg [byte]	1/1 1/1
PCMPGTD <i>Pack Compare Greater Than Dword</i> MMX Register 2 to MMX Register1 Memory with MMX Register	0F66 [11 mm1 mm2] 0F66 [mod mm r/m]	MMX reg 1 [dword] <--FFFF FFFFh-- if MMX reg 1 [dword] > MMX reg 2 [dword] MMX reg 1 [dword]<--0000 0000h--if MMX reg 1 [dword]NOT > MMX reg 2 [dword] MMX reg [dword] <--FFFF FFFFh-- if memory[dword] > MMX reg [dword] MMX reg [dword] <--0000 0000h-- if memory[dword] NOT > MMX reg [dword]	1/1 1/1

Table 6-25. 6x86MX Processor MMX Instruction Set Clock Count Summary (Continued)

MMX INSTRUCTIONS	OPCODE	OPERATION	CLOCK COUNT LATENCY/ THROUGHPUT
PCMPGTW <i>Pack Compare Greater Than Word</i> MMX Register 2 to MMX Register1	0F65 [11 mm1 mm2]	MMX reg 1 [word] <--FFFFh-- if MMX reg 1 [word] > MMX reg 2 [word]	1/1
Memory with MMX Register	0F65 [mod mm r/m]	MMX reg 1 [word]<--0000h-- if MMX reg 1 [word] NOT > MMX reg 2 [word] MMX reg [word] <--FFFFh-- if memory[word] > MMX reg [word] MMX reg [word] <--0000h-- if memory[word] NOT > MMX reg [word]	1/1
PMADDWD <i>Packed Multiply and Add</i> MMX Register 2 to MMX Register 1 Memory to MMX Register	0FF5 [11 mm1 mm2] 0FF5 [mod mm r/m]	MMX reg 1 [dword] <--add-- [dword]<---- MMX reg 1 [sign word]*MMX reg 2[sign word] MMX reg 1 [dword] <--add-- [dword] <---- memory[sign word] * Memory[sign word]	2/1 2/1
PMULHW <i>Packed Multiply High</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FE5 [11 mm1 mm2] 0FE5 [mod mm r/m]	MMX reg 1 [word] <--upper bits-- MMX reg 1 [sign word] * MMX reg 2 [sign word] MMX reg 1 [word] <--upper bits-- memory [sign word] * Memory [sign word]	2/1 2/1
PMULLW <i>Packed Multiply Low</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FD5 [11 mm1 mm2] 0FD5 [mod mm r/m]	MMX reg 1 [word] <--lower bits-- MMX reg 1 [sign word] * MMX reg 2 [sign word] MMX reg 1 [word] <--lower bits-- memory [sign word] * Memory [sign word]	2/1 2/1
POR <i>Bitwise OR</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FEB [11 mm1 mm2] 0FEB [mod mm r/m]	MMX Reg 1 [qword] <--logic OR-- MMX Reg 1 [qword], MMX Reg 2 [qword] MMX Reg [qword] <--logic OR-- MMX Reg [qword], memory[qword]	1/1 1/1
PSLLD <i>Packed Shift Left Logical Dword</i> MMX Register 1 by MMX Register 2 MMX Register by Memory MMX Register by Immediate	0FF2 [11 mm1 mm2] 0FF2 [mod mm r/m] 0F72 [11 110 mm] #	MMX reg 1 [dword] <--shift left, shifting in zeroes by MMX reg 2 [dword]-- MMX reg [dword] <--shift left, shifting in zeroes by memory[dword]-- MMX reg [dword] <--shift left, shifting in zeroes by [im byte]--	1/1 1/1 1/1
PSLLQ <i>Packed Shift Left Logical Qword</i> MMX Register 1 by MMX Register 2 MMX Register by Memory MMX Register by Immediate	0FF3 [11 mm1 mm2] 0FF3 [mod mm r/m] 0F73 [11 110 mm] #	MMX reg 1 [qword] <--shift left, shifting in zeroes by MMX reg 2 [qword]-- MMX reg [qword] <--shift left, shifting in zeroes by[qword]-- MMX reg [qword] <--shift left, shifting in zeroes by[im byte]--	1/1 1/1 1/1
PSLLW <i>Packed Shift Left Logical Word</i> MMX Register 1 by MMX Register 2 MMX Register by Memory MMX Register by Immediate	0FF1 [11 mm1 mm2] 0FF1 [mod mm r/m] 0F71 [11 110mm] #	MMX reg 1 [word] <--shift left, shifting in zeroes by MMX reg 2 [word]-- MMX reg [word] <--shift left, shifting in zeroes by memory[word]-- MMX reg [word] <--shift left, shifting in zeroes by[im byte]--	1/1 1/1 1/1
PSRAD <i>Packed Shift Right Arithmetic Dword</i> MMX Register 1 by MMX Register 2 MMX Register by Memory MMX Register by Immediate	0FE2 [11 mm1 mm2] 0FE2 [mod mm r/m] 0F72 [11 100 mm] #	MMX reg 1 [dword] <--arith shift right, shifting in zeroes by MMX reg 2 [dword]-- MMX reg [dword] <--arith shift right, shifting in zeroes by memory[dword]-- MMX reg [dword] <--arith shift right, shifting in zeroes by [im byte]--	1/1 1/1 1/1

Table 6-25. 6x86MX Processor MMX Instruction Set Clock Count Summary (Continued)

MMX INSTRUCTIONS	OPCODE	OPERATION	CLOCK COUNT LATENCY/ THROUGHPUT
PSRAW <i>Packed Shift Right Arithmetic Word</i> MMX Register 1 by MMX Register 2 MMX Register by Memory MMX Register by Immediate	0FE1 [11 mm1 mm2] 0FE1 [mod mm r/m] 0F71 [11 100 mm] #	MMX reg 1 [word] <--arith shift right, shifting in zeroes by MMX reg 2 [word]-- MMX reg [word] <--arith shift right, shifting in zeroes by memory[word--] MMX reg [word] <--arith shift right, shifting in zeroes by [im byte]--	1/1 1/1 1/1
PSRLD <i>Packed Shift Right Logical Dword</i> MMX Register 1 by MMX Register 2 MMX Register by Memory MMX Register by Immediate	0FD2 [11 mm1 mm2] 0FD2 [mod mm r/m] 0F72 [11 010 mm] #	MMX reg 1 [dword] <--shift right, shifting in zeroes by MMX reg 2 [dword]-- MMX reg [dword] <--shift right, shifting in zeroes by memory[dword]-- MMX reg [dword] <--shift right, shifting in zeroes by [im byte]--	1/1 1/1 1/1
PSRLQ <i>Packed Shift Right Logical Qword</i> MMX Register 1 by MMX Register 2 MMX Register by Memory MMX Register by Immediate	0FD3 [11 mm1 mm2] 0FD3 [mod mm r/m] 0F73 [11 010 mm] #	MMX reg 1 [qword] <--shift right, shifting in zeroes by MMX reg 2 [qword] MMX reg [qword] <--shift right, shifting in zeroes by memory[qword] MMX reg [qword] <--shift right, shifting in zeroes by [im byte]	1/1 1/1 1/1
PSRLW <i>Packed Shift Right Logical Word</i> MMX Register 1 by MMX Register 2 MMX Register by Memory MMX Register by Immediate	0FD1 [11 mm1 mm2] 0FD1 [mod mm r/m] 0F71 [11 010 mm] #	MMX reg 1 [word] <--shift right, shifting in zeroes by MMX reg 2 [word] MMX reg [word] <--shift right, shifting in zeroes by memory[word] MMX reg [word] <--shift right, shifting in zeroes by imm[word]	1/1 1/1 1/1
PSUBB <i>Subtract Byte With Wrap-Around</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FF8 [11 mm1 mm2] 0FF8 [mod mm r/m]	MMX reg 1 [byte] <---- MMX reg 1 [byte] subtract MMX reg 2 [byte] MMX reg [byte] <---- MMX reg [byte] subtract memory [byte]	1/1 1/1
PSUBD <i>Subtract Dword With Wrap-Around</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FFA [11 mm1 mm2] 0FFA [mod mm r/m]	MMX reg 1 [dword] <---- MMX reg 1 [dword] subtract MMX reg 2 [dword] MMX reg [dword] <---- MMX reg [dword] subtract memory [dword]	1/1 1/1
PSUBSB <i>Subtract Byte Signed With Saturation</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FE8 [11 mm1 mm2] 0FE8 [mod mm r/m]	MMX reg 1 [sign byte] <--sat-- MMX reg 1 [sign byte] subtract MMX reg 2 [sign byte] MMX reg [sign byte] <--sat-- MMX reg [sign byte] subtract memory [sign byte]	1/1 1/1
PSUBSW <i>Subtract Word Signed With Saturation</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FE9 [11 mm1 mm2] 0FE9 [mod mm r/m]	MMX reg 1 [sign word] <--sat-- MMX reg 1 [sign word] subtract MMX reg 2 [sign word] MMX reg [sign word] <--sat-- MMX reg [sign word] subtract memory [sign word]	1/1 1/1
PSUBUSB <i>Subtract Byte Unsigned With Saturation</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FD8 [11 mm1 mm2] 0FD8 [11 mm reg]	MMX reg 1 [byte] <--sat-- MMX reg 1 [byte] subtract MMX reg 2 [byte] MMX reg [byte] <--sat-- MMX reg [byte] subtract memory [byte]	1/1 1/1
PSUBUSW <i>Subtract Word Unsigned With Saturation</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FD9 [11 mm1 mm2] 0FD9 [11 mm reg]	MMX reg 1 [word] <--sat-- MMX reg 1 [word] subtract MMX reg 2 [word] MMX reg [word] <--sat-- MMX reg [word] subtract memory [word]	1/1 1/1
PSUBW <i>Subtract Word With Wrap-Around</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FF9 [11 mm1 mm2] 0FF9 [mod mm r/m]	MMX reg 1 [word] <---- MMX reg 1 [word] subtract MMX reg 2 [word] MMX reg [word] <---- MMX reg [word] subtract memory [word]	1/1 1/1

Table 6-25. 6x86MX Processor MMX Instruction Set Clock Count Summary (Continued)

MMX INSTRUCTIONS	OPCODE	OPERATION	CLOCK COUNT LATENCY/ THROUGHPUT
PUNPCKHBW <i>Unpack High Packed Byte</i> Data to Packed Words MMX Register 2 to MMX Register1 Memory to MMX Register	0F68 [11 mm1 mm2] 0F68 [11 mm reg]	MMX reg 1 [byte] <--interleave-- MMX reg 1 [up byte], MMX reg 2 [up byte] MMX reg [byte] <--interleave-- memory [up byte], MMX reg [up byte]	1/1 1/1
PUNPCKHDQ <i>Unpack High Packed Dword</i> Data to Qword MMX Register 2 to MMX Register1 Memory to MMX Register	0F6A [11 mm1 mm2] 0F6A [11 mm reg]	MMX reg 1 [dword] <--interleave-- MMX reg 1 [up dword], MMX reg 2 [up dword] MMX reg [dword] <--interleave-- memory [up dword], MMX reg [up dword]	1/1 1/1
PUNPCKHWD <i>Unpack High Packed Word</i> Data to Packed Dwords MMX Register 2 to MMX Register1 Memory to MMX Register	0F69 [11 mm1 mm2] 0F69 [11 mm reg]	MMX reg 1 [word] <--interleave-- MMX reg 1 [up word], MMX reg 2 [up word] MMX reg [word] <--interleave-- memory [up word], MMX reg [up word]	1/1 1/1
PUNPCKLBW <i>Unpack Low Packed Byte</i> Data to Packed Words MMX Register 2 to MMX Register1 Memory to MMX Register	0F60 [11 mm1 mm2] 0F60 [11 mm reg]	MMX reg 1 [word] <--interleave-- MMX reg 1 [low byte], MMX reg 2 [low byte] MMX reg [word] <--interleave-- memory [low byte], MMX reg [low byte]	1/1 1/1
PUNPCKLDQ <i>Unpack Low Packed Dword</i> Data to Qword MMX Register 2 to MMX Register1 Memory to MMX Register	0F62 [11 mm1 mm2] 0F62 [11 mm reg]	MMX reg 1 [word] <--interleave-- MMX reg 1 [low dword], MMX reg 2 [low dword] MMX reg [word] <--interleave-- memory [low dword], MMX reg [low dword]	1/1 1/1
PUNPCKLWD <i>Unpack Low Packed Word</i> Data to Packed Dwords MMX Register 2 to MMX Register1 Memory to MMX Register	0F61 [11 mm1 mm2] 0F61 [11 mm reg]	MMX reg 1 [word] <--interleave-- MMX reg 1 [low word], MMX reg 2 [low word] MMX reg [word] <--interleave-- memory [low word], MMX reg [low word]	1/1 1/1
PXOR <i>Bitwise XOR</i> MMX Register 2 to MMX Register1 Memory to MMX Register	0FEF [11 mm1 mm2] 0FEF [11 mm reg]	MMX Reg 1 [qword] <--logic exclusive OR-- MMX Reg 1 [qword], MMX Reg 2 [qword] MMX Reg [qword] <--logic exclusive OR-- memory[qword], MMX Reg [qword]	1/1 1/1

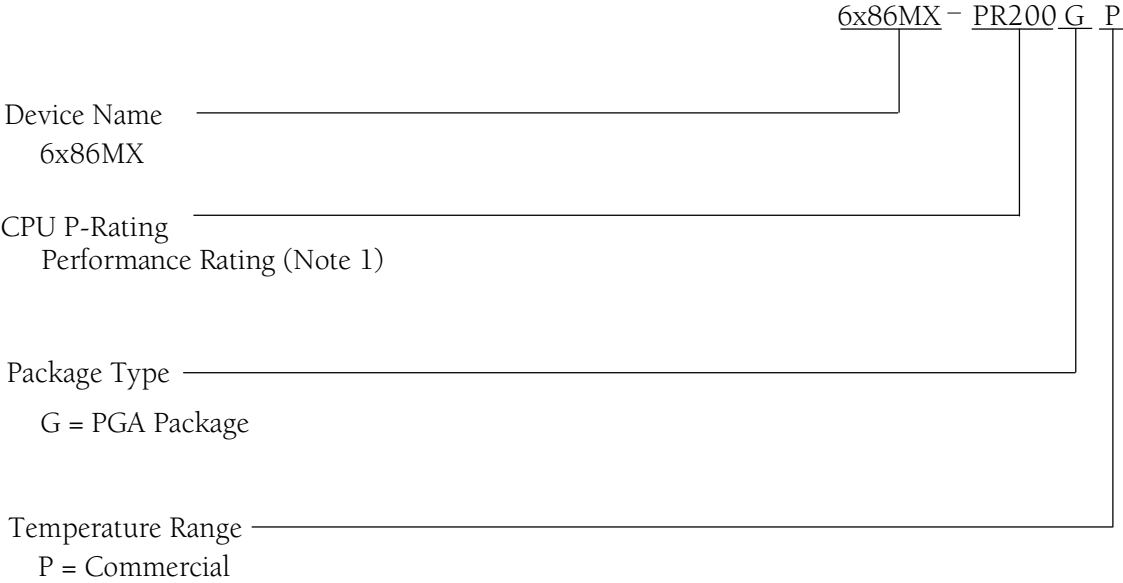
6x86MX™ PROCESSOR

Enhanced Sixth-Generation CPU
Compatible with MMX™ Technology



Appendix

Ordering Information



1740001

Note: For further information concerning Performance Ratings, visit our website at www.cyril.com.



The Cyrix 6x86MX CPU part numbers are listed below.

Cyrix 6x86MX™ Part Numbers

PART NUMBER	CLOCK MULTIPLIER	FREQUENCY (MHz)	
		BUS	INTERNAL
6x86MX - PR166GP	2.0	66	133
6x86MX - PR166GP	2.5	60	150
6x86MX - PR200GP	2.5	66	166
6x86MX - PR233GP	2.5	75	188
6x86MX - PR233GP	3.0	66	200
6x86MX - PR266GP	3.0	75	225
6x86MX - PR266GP	3.5	66	233

INDEX

'1+4' Burst Read Cycle	3-33	Cache Organization	2-58
A		Cache Units	1-14
AC Characteristics	4-6	Caches, Memory	2-57
Address Bus Signals	3-9	CCR0 Bit Definitions	2-26
Address Parity Signals	3-10	CCR1 Bit Definitions	2-27
Address Region Registers (ARRx)	2-33	CCR2 Bit Definitions	2-28
Address Space	2-47	CCR3 Bit Definitions	2-29
Architecture Overview	1-1	CCR4 Bit Definitions	2-30
B		CCR5 Bit Definitions	2-31
Back-Off Timing	3-47	CCR6 Bit Definitions	2-32
Base Field, Instruction Format	6-10	Clock Control Signals	3-7
Branch Control	1-13	Clock Count for CPU Instructions	6-14
Burst Cycle Address Sequence	3-32	Clock Count for FPU Instructions	6-31
Burst Write Cycles	3-35	Clock Count for MMX Instructions	6-38
Bus Arbitration	3-16	Clock Specifications	4-8
Bus Arbitration	3-44	Configuration Control Registers	2-24
Bus Cycle Control Signals	3-13	Control Registers	2-13
Bus Cycle Definition	3-11	Counter Event Control Register	2-40
Bus Cycle Types Table	3-12	CPUID Instruction	6-11
Bus Cycles, Non-pipelined	3-27	Cyrix Enhanced SMM Mode	2-78
Bus Hold, Signal States During	3-17	D	
Bus Interface	3-1	Data Bus Signals	3-10
Bus Interface Unit	1-17	Data Bypassing	1-12
Bus State Definition	3-24	Data Forwarding	1-9
Bus State Diagram for 6x86MX	3-25	Data Parity Signals	3-10
Bus Timing	3-23	DC Characteristics	4-4
C		Debug Register	2-44
Cache Coherency Signals	3-18	Descriptors	2-17
Cache Control Signals	3-14	Differences Between	
Cache Control Timing	3-41	6x86MX and 6x86 Processors	1-2
Cache Disable	2-39	E	
Cache Disable Region	2-37	Electrical Specifications	4-1
Cache Enable Region	2-37	Error Codes	2-69
Cache Inquiry Cycles	3-48	Event Type Register	2-41
Cache Inquiry Cycles, SMM Mode	3-54	EWBE# Timing	3-43
		Exceptions	2-62
		Exceptions in Real Mode	2-68

F

Flags Register	2-9
Floating Point Unit	1-17
FPU Error Interface	3-19
FPU Error Interface Signals	3-19
FPU Operations	2-86
Functional Blocks	1-3

G

Gate Descriptors	2-20
Gates, Protection Level Transfer	2-84

I

I/O Address Space	2-48
Index Field, Instruction Format	6-9
Initialization and Protected Mode	2-84
Initialization of the CPU	2-1
Input Hold Times	4-11
Input Setup Times	4-11
Inquiry Cycles Using AHOLD	3-51
Inquiry Cycles Using BOFF#	3-50
Inquiry Cycles Using HOLD/HLDA	3-49
Instruction Fields, General	6-2
Instruction Line Cache	1-15
Instruction Pointer Register	2-9
Instruction Set Overview	2-3
Instruction Set Summary	6-1
Instruction Set Tables	6-12
Instruction Set Tables Assumptions	6-12
Integer Unit	1-4
Interrupt Acknowledge Cycles	3-39
Interrupt and Exception Priorities	2-66
Interrupt Control Signals	3-13
Interrupt Vectors	2-64
Interrupts and Exceptions	2-62

J

JTAG AC Specifications	4-13
JTAG Interface	3-22

L

Lock Prefix	2-3
-------------	-----

M

Maximum Ratings, Absolute	4-2
Mechanical Specifications	5-1
Memory Addressing	2-50
Memory Addressing Methods	2-48
Memory Management Unit	1-16
MESI States, Unified Cache	2-57
MMX Operations	2-89
mod and r/m Fields, Inst. Format	6-6
Mode State Diagram	2-81
Model Specific Registers	2-38

N

NC and Reserved Pins	4-2
Non-pipelined Burst Read Cycles	3-30
Non-pipelined Bus Cycles	3-27

O

Offset Mechanism	2-49
Opcode Field, Instruction Format	6-4
Out-of-order Processing	1-5
Output Float Delays	4-10
Output Valid Delays	4-9

P

Package, Mechanical Drawing	5-5
Paging Mechanisms (Detail)	2-52
Performance Monitoring	2-38
Performance Monitoring Event Type	2-41
Pin Diagram, 296-Pin SPGA Package	5-1
Pin List, Sorted by Pin Number	5-3
Pin List, Sorted by Signal Name	5-4
Pipeline Stages	1-5
Pipelined Back-to-Back R/W Cycles	3-38
Pipelined Bus Cycles	3-36
Power and Ground Connections	4-1
Power Dissipation	4-5
Power Management Interface Signals	3-19

Power Management Interface Timing	3-60	SMM Operation	2-76
Prefix Field, Instruction Format	6-3	Speculative Execution	1-14
Privilege Level, Requested	2-8	ss Field, Instruction Format	6-9
Privilege Levels	2-82	Stopping the Input Clock	3-62
Programming Interface	2-1	Suspend Mode Signal States Table	3-21
Protected Mode Address Calculation	2-50	Suspend Mode, HALT Initiated	3-61
Protection, Segment and Page	2-82	System Management Mode (SMM)	2-70
Pull-Up and Pull-Down Resistors	4-1		
R		T	
RAW Dependency Example	1-10	Task Register	2-21
Recommended Operating Conditions	4-3	Test Registers	2-46
reg Field, Instruction Format	6-7	Thermal Characteristics	5-7
Region Control Registers (RCR _x)	2-36	Time Stamp Counter	2-38
Register Renaming	1-6	Timing, Bus	3-23
Register Sets	2-4	Translation Lookaside Buffer	2-52
Registers, Control	2-13	Translation Lookaside Buffer Testing	2-54
Registers, General Purpose	2-5	U	
Registers, 6x86MX Configuration	2-24	Unified Cache	1-14
Registers, System Set	2-11	Unified Cache Testing	2-58
Requested Privilege Level	2-8	V	
Reset Control Signals	3-7	Virtual 8086 Mode	2-85
RESET Timing	3-23	W	
S		WAR Dependency Example	1-7
Scratchpad Memory Locking	2-61	WAW Dependency Example	1-8
Segment Registers	2-7	Weak Locking	2-37
Selector Mechanism	2-51	Write Gathering	2-37
Selectors	2-7	Write Through	2-37
Shutdown and Halt	2-80		
Signal Description Table	3-2		
Signal Groups	3-1		
SL-Compatible SMM Mode	2-78		
SMHR Register	2-74		
SMI# Interrupt Timing	3-40		
SMM Instructions	2-75		
SMM Memory Space	2-71		
SMM Memory Space Header	2-72		

Cyrix Worldwide Offices

United States

Corporate Office

Richardson, Texas

Tel: (972) 968-8388

Fax: (972) 699-9857

Tech Support and Sales: (800) 462-9749

Internet: tech_support@cyrix.com

BBS: (972) 968-8610 (up to 28.8K baud)

See us on the Internet Worldwide Web:

www.cyrix.com

Europe

United Kingdom

Cyrix International Ltd.

Tel: +44 (0) 1 793 417777

Fax: +44 (0) 1 793 417770

Japan

Cyrix K.K.

Tel: 81-45-471-1661

Fax: 81-45-471-1666

Taiwan

Cyrix International, Inc.

Tel: 886-2-718-4118

Fax: 886-2-719-5255

Hong Kong

Cyrix International, Inc.

Tel: (852) 2485-2285

Fax: (852) 2485-2920

Cyrix Corporation

P.O. Box 850118

Richardson, TX 75085-0118

Tel: (972) 968-8388

Fax: (972) 699-9857

Order Number: 94xxx-xx - January 1997

Printed in the USA on recycled paper.