Cyrix SMM Programmer's Guide

Revision 2.1

1996
Order Number 94211-01

Cyrix Corporation
2703 North Central Expressway
Richardson, Texas 75080
United States of America

Order Number 94211-01 March 1996

Cyrix®
Advancing the Standards

TABLE OF CONTENTS

1
2
3
4
A
B
C

[redacted]

Cyrix®
Advancing the Standards

1. SMM OVERVIEW

1.1 Introduction

This Programmer's Guide is provided to assist programmers in the creation of software that uses the Cyrix® System Management Mode (SMM) for the following Cyrix products:

- Cx486DX2™ processor
- Cx486DX4™ processor
- 5x86™ processor
- 6x86™ processor

Note: "6x86" is a product code that will be replaced by a product name at a later date. This guide should be used in conjunction with the appropriate Cyrix Processor Data Book.

This manual is an update to the 1992 *Cx486SLC/e SMM Programmer's Guide* that describes SMM operations for the Cx486SLC/e and Cx486DX Cyrix CPUs.

SMM provides the system designer with another operating mode for the CPU. Within this document, the standard x86 operating modes (real, V86, and protected) are referred to as normal mode. Normal-mode operation can be interrupted by an SMI interrupt or special instruction that places the processor in System Management Mode (SMM). SMM can be used to enhance the functionality of the system by providing power management, register shadowing, peripheral emulation and other system-level functions. SMM can be totally transparent to all software, including protected-mode operating systems.

1.2 Cyrix SMM Features

The Cyrix microprocessors have programmable location and size for the SMM memory region. The CPUs automatically save minimal register information, reducing the time needed for SMM entry and exit. The SMM implementation by Cyrix provides unique instructions that save additional segment registers. The x86 MOV instruction can be used to save the general purpose registers.

The Cyrix processors simplify I/O trapping by providing I/O type identification and instruction restarting. Cyrix CPUs also make available to the SMM routine information that can simplify peripheral register shadowing.

Cyrix provides a method to prevent SMM configuration registers from being accessed by applications. Not allowing an application to disable or alter SMM operation is useful for anti-virus or security measures.

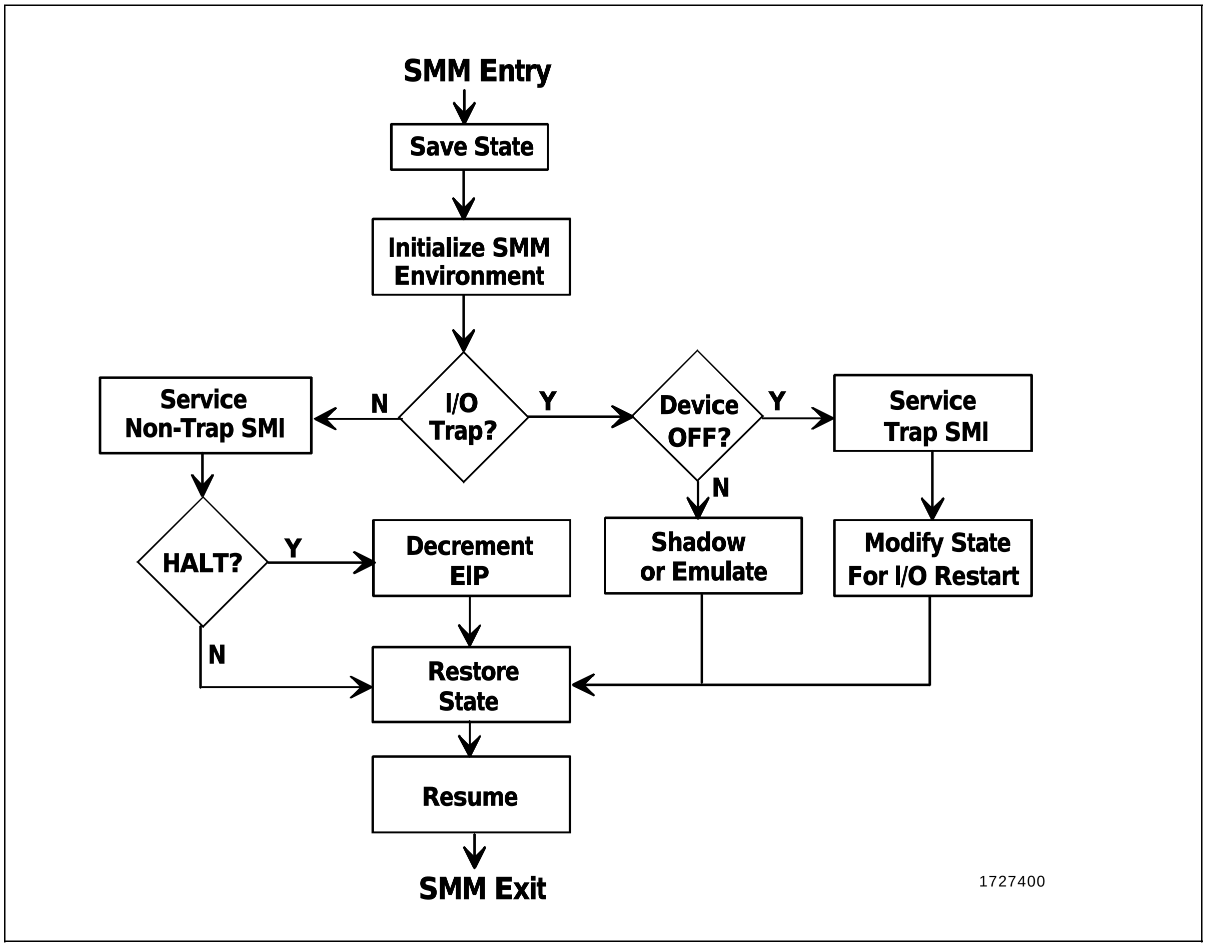

Figure 1-1. Typical SMM Routine

1.3 Typical SMM Routines

A typical SMM routine is illustrated in the flowchart shown in Figure 1-1. Upon entry to SMM, the CPU registers that will be used by the SMM routine must be saved. The SMM environment is initialized by setting up an Interrupt Descriptor Table, initializing segment limits, and setting up a stack. If entry to SMM results from an I/O bus cycle, the SMM routine can monitor peripheral activity, shadow read-only ports, and emulate peripherals in software. If a peripheral is powered down, the SMM routine can power it up and reissue the I/O instruction. If the SMM routine is not the result of an I/O bus cycle, non-trap SMI functions can be serviced. If an HLT instruction is interrupted by an SMI then the HLT instruction should be restarted when the SMM routine is completed. Before normal operation is resumed, any CPU registers modified during the SMM routine must be restored to their previous state.

2. SMM IMPLEMENTATION

This chapter describes the Cyrix SMM System interface. SMM operations for Cyrix microprocessors are similar to related operations performed by other x86 microprocessors.

Cyrix CPUs support two SMM modes, Cyrix SMM mode and SL SMM mode—except for the 6x86 which supports only SL SMM mode.

The CPU defaults to Cyrix SMM mode. Setting SMM_MODE bit (in CCR3) will cause the CPU to operate in SL SMM mode.

2.1 SMM Pins

In either SMM mode, two unique pins are required to support SMM. These pins perform three functions:

1. Signaling when an SMI interrupt should occur,
2. Informing the chipset that the CPU is in SMM mode,
3. Informing the chipset whether the bus cycle is intended for SMM memory or system memory.

Signals at the SMI# and SMADS# pins are used to implement SMM.

2.2 Cyrix SMM Mode

The CPU defaults to Cyrix SMM mode. Cyrix SMM mode is not supported by the 6x86.

An SMM routine can be started by asserting the SMI# "input" pin. Once the SMM routine has begun, the SMI# pin becomes an output pin that signals the chip set that an SMM routine is in progress. The SMADS# address strobe signal is generated (instead of an ADS# address strobe signal) while the CPU is executing instructions or accessing data in SMM address space

2.2.1 SMI# Pin Timing

To enter Cyrix SMM mode, the SMI# pin must be asserted for at least one CLK period (two clocks if SMI# is asserted asynchronously). To accomplish I/O trapping, the SMI# signal should be asserted two clocks before the RDY# for that I/O cycle. Once the CPU recognizes the active SMI# input, the CPU drives the SMI# input low for the duration of the SMM routine.

The SMM routine is terminated with an SMM-specific resume instruction (RSM). When the RSM instruction is executed, the CPU drives the SMI# pin high for one CLK period. The SMI# pin must be allowed to go high for one CLK at the end of the SMM routine to allow for the next SMI to be recognized. Since the SMI# pin is bi-directional, only one SMI# interrupt can become active at one time.

2.2.2 Address Strobes

The CPU has two address strobes, ADS# and SMADS#. ADS# is the address strobe used during normal operations. The SMADS# address strobe replaces ADS# during SMM for memory accesses when data is written, read, or fetched in the SMM defined region. Using a separate address strobe simplifies chipset design.

During an SMM interrupt routine, control can be transferred to main memory via a JMP, CALL, Jcc instruction, or by execution of a software interrupt (INT), or execution of a hardware interrupt (INTR or NMI).

Code accesses in main memory will assert ADS#. ADS# will also be asserted for data accesses outside of the defined SMM address region. It is assumed, but not required, that the chipset ultimately translates SMADS# and a particular address to some other address.

To access data in main memory that overlaps the SMM address space, the MMAC bit (CCR1, bit 3) must be set. This allows ADS# strobes to be generated for data accesses in memory that overlap SMM memory while in SMM mode. While in SMM mode it is not possible to execute code in main memory that overlaps SMM space.

SMADS# can also be generated for memory reads, memory writes, and code fetches within the defined SMM region when the SMAC bit (CCR1, bit 2) is set while in normal mode. The generation of SMADS# permits a program in normal mode to execute out of SMM memory. The RSM instruction should not be executed when not servicing an SMM interrupt unless valid return information is first written into the SMM header.

2.2.3 Cache Coherency

SMM memory is never cached in the CPU internal cache. This makes cache coherency completely transparent to the SMM programmer using Cyrix SMM mode. If the CPU cache is in write-back mode, all write-back cycles will be directed to normal memory with the use of the ADS# signal. An INVD or WBINVD will write dirty data out to normal memory even if it overlaps with SMM space.

SMM memory can be cached by an external cache controller, but it is up to the cache designer to be sure to maintain a distinction between SMM memory space and normal memory space.

The A20M# input to the CPU is ignored for all SMM space accesses (that is, any access that uses SMADS#).

2.3 SL SMM Mode

SL SMM mode is selected by the SMM_MODE bit in CCR3 The 6x86 supports only SL SMM mode.

The SMI# and SMADS# pins are used to implement SL SMM Mode. (SMADS# is referred to as SMIACT# on the 6x86.) The SMI# pin is an input pin used by the chipset to signal the CPU that an SMI has been requested. While the CPU is in the process of servicing an SMI interrupt, the SMADS# (SMIACT#) pin is an output used to signal the chipset that the SMM processing is occurring. The ADS# address strobe signal is asserted in order to access data in either normal memory or SMM address space.

2.3.1 SMI# Input

SMI# is an edge-triggered input pin sampled by two rising edges of CLK. SMI# must meet certain setup and hold times to be detected on a specific clock edge. To accomplish I/O trapping, the SMI# signal should be asserted three clocks before the RDY# or BRDY# for that I/O cycle.   Once the CPU recognizes the active SMI# input, the CPU drives SMADS# (SMIACT#) active for the duration of the SMM routine. The SMM routine is terminated with an SMM-specific resume instruction (RSM). When the RSM instruction is executed, the CPU negates the SMADS# (SMIACT#) pin after the last bus cycle to SMM memory. While executing the SMM service routine, one additional SMI# can be latched for service after resuming from the first SMI.

2.3.2 SMADS# (SMIACT#) Address Strobe

The CPU uses one address strobe, ADS#, to initiate memory cycles for both normal and SMM memory.

The chipset must monitor the address on the bus to determine if a given cycle is intended for normal or SMM memory. If SMADS# (SMIACT#) is inactive when an ADS# is asserted, the cycle will access normal memory. If SMADS# (SMIACT#) is active when an ADS# is asserted, the chipset must compare the address bus to the address range for SMM memory. If the address is within the SMM address region, the cycle should be directed to SMM memory. If the address is outside of the SMM address region, the cycle should be directed to normal memory.

Normal memory located within the same physical address range as the SMM address region can only be accessed from within SMM mode by chipset-specific functions which will relocate the normal memory to an address that is accessible to the SMM code. In normal mode, SMM memory can be initialized by using chipset-specific functions to map the SMM memory into normal memory so that it can be accessed.

The MMAC and SMAC bits in CCR1 should not be used while in SL SMM mode. See Appendix C for details on how these bits function in each of the Cyrix CPUs.

2.3.3 Cache Coherency

Intel's SL Enhanced 486 allows SMM memory accesses to be cached. This may cause coherency problems in systems where SMM memory space and normal memory space overlap. Therefore, Intel recommends one of two options: (1) flush the cache when entering and exiting an SMM service routine, or (2) flush the cache when entering an SMM service routine and then make all SMM accesses non-cacheable using the KEN# pin. In both cases, Intel recommends asserting the FLUSH# input when SMIACT# is asserted. This is acceptable for a CPU with a write-through cache because the flush invalidates the cache in a single clock.

Therefore, the Cyrix CPU must also write back and invalidate the cache prior to asserting SMADS# (SMIACT#). No dirty data can exist in the CPU (cache and write buffers) at the time that SMADS# (SMIACT#) is asserted. On the 486DX2/DX4 this flush is done automatically before SMADS# (SMIACT#) is asserted.

On 5x86 and 6x86 CPUs, the chipset must drive FLUSH# on the same clock as SMI# to ensure that the dirty data is written out to memory before the SMIACT# is asserted.

If the software instruction SMINT is used to enter SMM a WBINVD instruction should be executed immediately before the SMINIT instruction to assure that no dirty data is in the cache.

A bus snoop will not hit in the CPU cache if the FLUSH# pin has been asserted before entering SMM. Cyrix CPUs prevent dirty data hits within SMM because the SMM space is always non-cacheable.

2.4 Configuration Control Registers and SMM

This section describes fields in the Configuration Registers that configure SMM operations. Fields not related to SMM are not described in this manual and are shown as blank fields in the configuration register tables. For a complete description of the configuration registers, refer to the appropriate data book.

All configuration-register bits related to SMM and power management are cleared to 0 when RESET is asserted. Asserting WM_RST does not affect the configuration registers.

These registers are accessed by writing the register index to I/O port 22h. I/O port 23h is used for data transfer. Each data transfer to I/O port 23h must be preceded by an I/O port 22h register-index selection, otherwise the port 23h access will be directed off chip.

Before accessing these registers, all interrupts must be disabled. A problem could occur if an interrupt occurs after writing to port 22h but before accessing port 23h. The interrupt service routine might access port 22h or 23h. After returning from the interrupt, the access to port 23h would be redirected to another index or possibly off chip.

An SMI interrupt cannot interrupt accesses to the configuration registers. After writing an index to port 22h in the CPU configuration space, SMI interrupts are disabled until the corresponding access to port 23h is complete.

The portions of the configuration registers that apply to SMM and power management are described in the following pages.

Table 2-1. CCR1 Register

Register INDEX = C1h

7	6	5	4	3	2	1	0
SM3				MMAC	SMAC	USE_SMI	

Table 2-2. CCR1 Bit Definitions

BIT POSITION	NAME	DESCRIPTION	Notes
1	USE_SMI	Enable SMM Pins. If = 1: The SMI# input/output pin and SMADS# (SMIACT#) output pin are enabled. USE_SMI must be set to 1 before any attempted access to SMM memory is made. If = 0: the SMI# input pin is ignored and SMADS# (SMIACT#) output pin floats. Execution of Cyrix specific SMM instructions will generate an invalid opcode exception.	Also called SMI
2	SMAC	System Management Memory Access. If = 1: SMI# input is ignored. Memory accesses while in normal mode that fall within the specified SMM address region generate an SMADS# (SMIACT#) output and access SMM memory. Instructions with SMM opcodes are enabled. If = 0: All memory accesses in normal mode go to system memory with ADS# output active. In normal mode, execution of Cyrix specific SMM instructions generate an invalid opcode exception.	Valid on Cx486DX2/DX4 and 5x86 only when operating in Cyrix SMM mode. SMAC is always available for 6x86.
3	MMAC	Main Memory Access. If = 1: Data accesses while in SMM mode that fall within the specified SMM address region will generate an ADS# output and access main memory. Code fetches are not effected by the MMAC bit. Code fetches from the SMM address region always generate an SMADS# output and access SMM memory. If both the SMAC and MMAC bits are set to 1, the MMAC bit has precedence. If = 0: All memory accesses to the SMM address region while in SMM mode go to SMM memory with SMADS# output active.	Not available for 6x86. Do not set MMAC unless operating in Cyrix SMM mode.
7	SM3	SMM Space Address Region 3 If = 1 Address Region 3 (ARR3) is redefined as the SMM Address Region (SMAR).	Available for 6x86 only.

Table 2-3. CCR2 Register

Register INDEX = C2h

7	6	5	4	3	2	1	0
USE_SUSP				SUSP_HALT			

Table 2-4. CCR2 Bit Definitions

BIT POSITION	NAME	DESCRIPTION	Notes
3	SUSP_HALT	Suspend on HALT. If = 1: CPU enters suspend mode following execution of a HLT instruction. If = 0: CPU does not enter suspend mode following execution of a HLT instruction.	Also called HALT.
7	USE_SUSP	Enable Suspend Pins. If = 1: SUSP# input and SUSPA# output are enabled. If = 0: SUSP# input is ignored and SUSPA# output floats.	Also called SUSP.

Table 2-5. CCR3 Register

INDEX = C3h

7	6	5	4	3	2	1	0
				SMM_MODE		NMI_EN	SMI_LOCK

Table 2-6. CCR3 Bit Definitions

BIT POSITION	NAME	DESCRIPTION	Notes
0	SMI_LOCK	SMM Register Lock. If = 1: the following Configuration Control Register bits can not be modified unless operating in SMM mode: USE_SMI, SMAC, MMAC, NMI_EN, SM3 and SMAR. If = 0: any program in normal mode, as well as SMM software, has access to all Configuration Control Registers. Once set, the SMI_LOCK bit can only be cleared by asserting the RESET pin.	
1	NMI_EN	NMI Enable. If = 1: NMI is enabled during SMM. This bit should only be set temporarily while in the SMM routine to allow NMI interrupts to be serviced. NMI_EN should not be set to 1 while in normal mode. If NMI_EN = 1 when an SMI occurs, an NMI could occur before the SMM code has initialized the Interrupt Descriptor Table. If = 0: NMI (Non-Maskable Interrupt) is not recognized during SMM. One occurrence of NMI can be latched and serviced after SMM mode is exited. The NMI_EN bit should be cleared before executing a RSM instruction to exit SMM.	Also called NMIEN
3	SMM_MODE	SMM Mode If = 1: SMM pins function as defined for SL-compatible mode. If = 0: SMM pins function as defined for Cyrix SMM compatible mode.	Not available on 6x86 as 6x86 operates in SL SMM mode only.

Table 2-7. SMM Address Region Registers (SMAR)

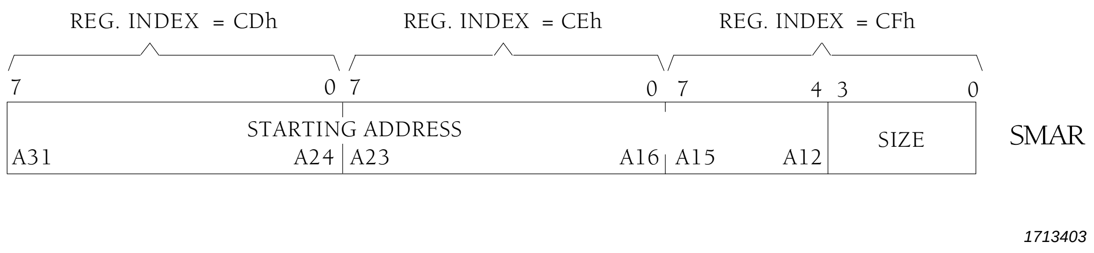

Table 2-8. SMAR Register SIZE Field

Bits 3-0	BLOCK SIZE
0h	Disable
1h	4 KBytes
2h	8 KBytes
3h	16 KBytes
4h	32 KBytes
5h	64 KBytes
6h	128 KBytes
7h	256 KBytes

Bits 3-0	BLOCK SIZE
8h	512 KBytes
9h	1 MBytes
Ah	2 MBytes
Bh	4 MBytes
Ch	8 MBytes
Dh	16 MBytes
Eh	32 MBytes
Fh	4 KBytes (same as 1h) 4 GBytes (6x86 only)

Note for 6x86 processors only: Address Region 3 (ARR3) is designated as SMM address space if CCR1 bit 7 (SM3) is set.

2.5 SMM Instruction Summary

Cyrix has added seven new instructions to the x86 standard instruction set to aid in SMM programming. These instructions are only valid when:

1) USE_SMI = 1
2) SMAR > 0
3) Current Privilege Level (CPL) = 0
4) SMAC bit is set or the CPU is in SMM mode

Note: There are minor differences between CPUs concerning when these instruction are valid as detailed in Appendix C.

The CPU will generate an invalid opcode fault when the conditions above are not met and one of the SMM instructions is executed. The assembly language macro SMIMAC.INC listed in Appendix A will automatically generate the appropriate machine code when included in a source file containing Cyrix SMM instructions.

Most of the Cyrix SMM instructions are used to access the non-programmer visible internal descriptors. The standard x86 instructions cannot access this information inside the CPU. This information is stored in memory in a 10-byte area that is comprised of both the descriptor (8 bytes) and the segment register/selector (2 bytes). The 8-byte descriptor is in the same format that it is found in the GDT or LDT. If the data area is dword aligned, the memory access time will be minimized.

Table 2-9. Register and Descriptor Save Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
SELECTOR or SEGMENT																+8
BASE 31-24								G	D	0	AVL	LIMIT 19-16				+6
P	DPL		DT	TYPE				BASE 23-16								+4
BASE 15-0																+2
LIMIT 15-0																+0

2.5.1 RSDC - Restore Register and Descriptor

Table 2-10. Restore Register and Descriptor

Instruction	Opcode	Parameters
RSDC	0F 79 [mod sreg3 r/m]	sreg3, mem80

RSDC loads the information at the mem80 into a segment register/selector and its associated descriptor. Attempting to use this instruction to load the Code Segment or Code Selector will generate an invalid opcode instruction. Code Segment or Code Selector is restored from the SMM header as part of the RSM instruction.

2.5.2 RSLDT - Restore LDT and Descriptor

Table 2-11. Restore LDT and Descriptor

Instruction	Opcode	Parameters
RSLDT	0F 7B [mod 000 r/m]	mem80

RSLDT loads the information at the mem80 into Local Descriptor Table Register and its associated descriptor.

2.5.3 RSM - Resume Back to Normal Mode

Table 2-12. Resume Back to Normal Mode

Instruction	Opcode	Parameters
RSM	0F AA	None

RSM will restore the state of the CPU from the SMM header at the top of SMM space and exit SMM. This is the last instruction executed in an SMM handler.

2.5.4 RSTS - Restore TSR and Descriptor

Table 2-13. Restore TSR and Descriptor

Instruction	Opcode	Parameters
RSTS	0F 7D [mod 000 r/m]	mem80

RSTS loads the information at the mem80 address into the Task Register and its associated descriptor.

2.5.5 SMINT - Software SMM Interrupt

Table 2-14. Software SMM Interrupt

Instruction	Opcode	Parameters
SMINT	0F 7E	None

SMINT will cause the CPU to enter SMM as though the hardware SMI# pin were sampled enabled. The SMINT instruction sets the "S" bit in the SMM header. The SMI# signal is not driven by the CPU if an SMM routine is entered using an SMINT instruction and if the CPU is operating in Cyrix SMM mode. If operating an 6x86 in write-back mode, a WBINVD instruction should be executed immediately proceeding a SMINT instruction to preserve cache coherency.

2.5.6 SVDC - Save Register and Descriptor

Table 2-15. Save Register and Descriptor

Instruction	Opcode	Parameters
SVDC	0F 78 [mod sreg3 r/m]	mem80, sreg3

SVDC saves the contents of a segment register/selector and its associated descriptor to memory at mem80. This instruction can be used on any segment/selector including the Code Segment.

2.3.7 SVLDT - Save LDT and Descriptor

Table 2-16. Save LDT and Descriptor

Instruction	Opcode	Parameters
SVLDT	0F 7A [mod 000 r/m]	mem80

SVLDT saves the Local Descriptor Table Selector and non-programmer visible descriptor information at the address location mem80.

2.3.8 SVTS - Save TSR and Descriptor

Table 2-17. Save TSR and Descriptor

Instruction	Opcode	Parameters
SVTS	0F 7C	mem80

SVTS saves the Task Register and its associated descriptor to address location mem80.

[redacted]

Cyrix®
Advancing the Standards

3. SMM SOFTWARE CONSIDERATIONS

This section provides information helpful in the development of SMM code.

3.1 Initializing SMM

Many systems have memory controllers that aid in the initialization of SMM memory. Cyrix SMM features allow the initialization of SMM memory without external hardware memory remapping.

When loading SMM memory with an SMM interrupt handler it is important that the SMI# does not occur before the handler is loaded.

To load SMM memory with a program it is first necessary to enable SMM memory without enabling the SMI pins. This is done by setting SMAC = 1 and loading SMAR with the SMM address region. Setting USE_SMI = 1 will then map the SMM memory region over main memory. The SMM region is physically mapped by the assertion of SMADS# to allow memory access within the SMM region. A REP MOV instruction can then be used to transfer the program to SMM memory. After initializing SMM memory, negate SMAC to activate potential SMI#s.

SMM space can be located anywhere in the 4-GByte address range. However, if the location of SMM space is above 1 MByte, the value in CS will truncate the segment above 16 bits when stored from the stack. This would prohibit doing calls or interrupts from real mode without restoring the 32-bit features of the 486 because of the incorrect return address on the stack.

```
; load SMM memory from system memory (Cyrix SMM mode only)

include SMIMAC.INC
SMMBASE = 68000h
SMMSIZE = 4000h    ;SMM SIZE is 16K
SMI   = 1 shl 1
SMAC  = 1 shl 2
MMAC  = 1 shl 3
;interrupts should be disabled here
      mov          al, 0cdh    ;index SMAR, SMM base<A31-A24>
      out          22h, al     ;select
      mov          al, 00h     ;set high SMM address to 00
      out          23h, al     ;write value
      mov          al, 0ceh    ;index SMAR,SMM base<A23-A16>
      out          22h, al     ;select
      mov          al, 06h     ;set mid SMM address to 06h
      out          23h, al     ;write value
      mov          al, 0cfh    ;SMAR,SMM base<A15-A12> & SIZE
      out          22h, al     ;select
      mov          al, 083h    ;set SMM lower addr. 80h, 16K
      out          23h, al     ;write value
      mov          al, 0c1h    ;index to CCR1
      out          22h, al     ;select CCR1 register
      in           al, 23h     ;read current CCR1 value
      mov          ah,  al     ;save it
      mov          al, 0c1h    ;index to CCR1
      out          22h, al     ;select CCR1 register
      mov          al, ah
      or           al, SMI or SMAC; set SMI and SMAC
      out          23h, al     ;new value now in CCR1, SMM now
                               ;mapped in
      mov          ax, SMMBASE shr 4
      mov          es, ax
      mov          edi, 0      ;es:di = start of the SMM area
      mov          esi, offset SMI_ROUTINE  ;start of copy of SMM
      mov          ax, seg SMI_ROUTINE     ;routine in main memory
      mov          ds, ax
      mov          ecx, (SMI_ROUTINE_LENGTH+3)/4  ;calc. length

; this line copies the SMM routine from DS:ESI to ES:EDI
      rep
      movs dword ptr es:[edi],dword ptr ds:[esi]

; now disable SMI by clearing SMAC and SMI
      mov          al, 0c1h    ;index to CCR1
      out          22h, al     ;select CCR1 register
      mov          al, ah      ;AH is still old value
      and          al, NOT SMAC;disable SMAC, enable SMI#
      out          23h, al     ;write new value to CCR1
```

3.2 SMM Handler Entry State

Before entering an SMM routine, certain portions of the CPU state are saved at the top of SMM memory. To optimize the speed of SMM entry and exit, the CPU saves the minimum CPU state information necessary for an SMI interrupt handler to execute and return to the interrupted context.

The information is saved to the SMM header at the top of the defined SMM region (starting at SMM base + size - 30h) as shown in Figure 3-1. Only the CS, EIP, EFLAGS, CR0, and DR7 are saved upon entry to SMM. Data accesses must use a CS segment override to save other registers and access data in SMM memory. To use any other segment register, the SMM programmer must first save the contents using the SVDC instruction for segment registers or MOV operations for general purpose registers (See Cyrix SMM instruction description Section 2.3). It is possible to save all the CPU registers as needed. See Section 3.3 for an example of saving and restoring the entire CPU state.

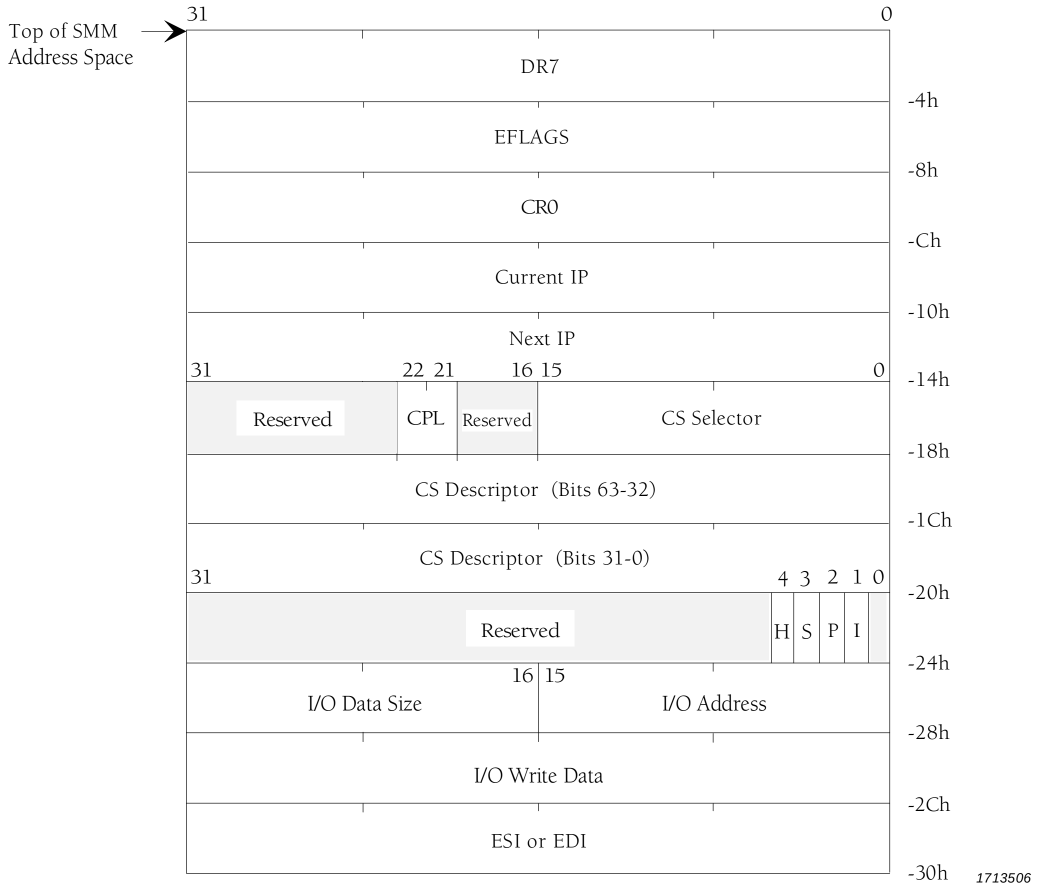

Figure 3-1. SMM Memory Space Header

Upon execution of an RSM instruction, control is returned to NEXT_IP. The value of NEXT_IP may need to be modified for restarting I/O instructions. This modification is a simple move of the CURRENT_IP value to the NEXT_IP location. Execution is then returned to the I/O instruction, rather than to the instruction after the I/O instruction.

This CURRENT_IP value is valid only if the instruction executing when the SMI occurred was an I/O instruction. Table 3-1 lists the SMM header information needed to restart an I/O instruction. The restarting of I/O instructions may also require modifications to the ESI, ECX and EDI depending on the instruction (see Section 3.6 for an example.)

Table 3-1. I/O Trap Information

Bit	Description	Size
H	HALT Indicator If = 1: The CPU was in a halt or shut down prior to serving the SMM interrupt. If = 0: The CPU was not in a halt or shut down prior to serving the SMM interrupt.	1 bit
S	Software SMM Entry Indicator S=1, if current SMM is the result of an SMINT instruction. S=0, if current SMM is not the result of an SMINT instruction.	1 bit
P	REP INSx/OUTSx Indicator If = 1: Current instruction does not have a REP prefix If = 0: Current instruction has a REP prefix	1 bit
I	IN, INSx, OUT, or OUTSx Indicator If = 1: Current instruction performed an I/O WRITE If = 0: Current instruction performed an I/O READ	1 bit
I/O Data Size	Indicates size of data for the trapped I/O 01h = byte 03h = word 0fh = dword	2 bytes
I/O Address	Address of the trapped I/O	2 bytes
I/O Write Data	Data written during I/O trapped write	4 bytes
ESI or EDI	Value of appropriate index register before the trapped I/O instruction	4 bytes

The EFLAGS, CR0 and DR7 registers are set to their reset values upon entry to the SMI handler. Resetting these registers has implications for setting breakpoints using the debug registers. Breakpoints in SMM address space can not be set prior to the SMI interrupt using debug registers. A debugger will only be able to set a code breakpoint using INT 3 outside of the SMM handler. See Section 3.11 for restrictions on debugging SMM code. Once the SMI has occurred and the debugger has control in SMM space, the debug registers can be used for the remainder of the SMI handler execution.

If the S bit in the SMM header is set, the SMM entry resulted from an SMINT instruction.

Upon SMM entry, I/O trap information is stored in the SMM memory space header. This information allows restarting of I/O instructions, as well as the easy emulation of I/O functions by the SMM handler. This data is valid only if the instruction executing when the SMI occurred was an I/O instruction. On DX2/DX4 devices, only I/O writes generate valid I/O fields to allow I/O restart. On 5x86 and 6x86 devices, both I/O reads and I/O write traps result in valid I/O fields and current P and I field values.

If the H bit in the SMM header is set, a HLT instruction was being executed when the SMI occurred. To resume execution of the HLT instruction, the field NEXT-IP in the SMM header should be decremental by one before executing RSM instruction. The DX2/DX4 processors do not support the H bit. Refer to Appendix B for instruction on handling resume to halt operations on a DX2 or DX4.

The values found in the I/O trap information fields are specified below for all cases.

Table 3-2. Valid I/O Trap Cases

Valid Cases	P	I	I/O Write Data Size	I/O Write Address	I/O Write Data	ESI or EDI
Not an I/O instruction	x	x	x	x	x	x
IN al	0	0	01h	I/O Address	xxxxxxx x	EDI
IN ax	0	0	03h	I/O Address	xxxxxxx x	EDI
IN eax	0	0	0Fh	I/O Address	xxxxxxx x	EDI
INSB	0	0	01h	I/O Address	xxxxxxx x	EDI
INSW	0	0	03h	I/O Address	xxxxxxx x	EDI
INSD	0	0	0Fh	I/O Address	xxxxxxx x	EDI
REP INSB	1	0	01h	I/O Address	xxxxxxx x	EDI
REP INSW	1	0	03h	I/O Address	xxxxxxx x	EDI
REP INSD	1	0	0Fh	I/O Address	xxxxxxx x	EDI
OUT al	0	1	01h	I/O Address	xxxxxxd d	ESI
OUT ax	0	1	03h	I/O Address	xxxxddd d	ESI
OUT eax	0	1	0Fh	I/O Address	ddddddd d	ESI
OUTSB	0	1	01h	I/O Address	xxxxxxd d	ESI
OUTSW	0	1	03h	I/O Address	xxxxddd d	ESI
OUTSD	0	1	0Fh	I/O Address	ddddddd d	ESI
REP OUTSB	1	1	01h	I/O Address	xxxxxxd d	ESI
REP OUTSW	1	1	03h	I/O Address	xxxxddd d	ESI
REP OUTSD	1	1	0Fh	I/O Address	ddddddd d	ESI

Note: x = invalid

Note: For DX2/DX4 devices, the I/O Data size, I/O address, I/O address, I/O data fields are not valid for IN instructions. The P, I and ESI or EDI fields are valid to allow I/O restart.

Upon SMM entry, the CPU enters the state described in Table 3-1.

Table 3-1. SMM Entry State

Register	Register Content	Comments
CS	SMM base specified by SMAR	CS limit is set to 4 GBytes (64 KBytes for a DX2/DX4 devices).
EIP	0000 0000h	Begins execution at the base of SMM memory
EFLAGS	0000 0002h	Reset State
CR0	0000 0010h	DX2/DX4 only: EM is not modified.
	6000 0010h	Other than DX2/DX4: NW will not be modified if LOCK_NW is set.
DR7	0000 0400h	Traps disabled

3.3 Maintaining the CPU State

The following registers are not automatically saved on SMM entry or restored on SMM exit.

General Purpose Registers:	EAX, EBX, ECX, EDX
Pointer and Index Registers:	EBP, ESI, EDI, ESP
Selector/Segment Registers:	DS, ES, SS, FS, GS
Descriptor Table Registers:	GDTR, IDTR, LDTR, TR
Control Registers:	CR2, CR3
Debug Registers:	DR0, DR1, DR2, DR3, DR6
Configuration Registers:	all valid configuration registers
FPU Registers:	Entire FPU state.

If the SMM routine will use any of these registers, their contents must be saved after entry into the SMM routine and then restored prior to exit from SMM. Additionally, if power is to be removed from the CPU and the system is required to return to the same system state after power is reapplied, then the entire CPU state must be saved to a non-volatile memory subsystem such as a hard disk.

3.3.1 Maintaining Common CPU Registers

The following is an example of the instructions needed to save the entire CPU state and restore it. This code sequence will work from real mode if the conditions needed to execute Cyrix SMM instructions are met (see Section 2.3). Configuration registers would also need to be saved if power is to be removed.

```
; Save and Restore the common CPU registers.
; The information automatically saved in the
; header on entry to SMM is not saved again.
include SMIMAC.INC

      .386P                    ;required for SMIMAC.INC macro
      mov     cs:save_eax,eax
      mov     cs:save_ebx,ebx
      mov     cs:save_ecx,ecx
      mov     cs:save_edx,edx
      mov     cs:save_esi,esi
      mov     cs:save_edi,edi
      mov     cs:save_ebp,ebp
      mov     cs:save_esp,esp
      svdc    cs:,save_ds,ds
      svdc    cs:,save_es,es
      svdc    cs:,save_fs,fs
      svdc    cs:,save_gs,gs
      svdc    cs:,save_ss,ss
      svldt   cs:,save_ldt      ;sldt is not valid in real mode
      svts    cs:,save_tsr      ;str is not valid in real mode
      db      66h               ;32bit version saves everything
      sgdt    fword ptr cs:[save_gdt]
      db      66h               ;32bit version saves everything
      sidt    fword ptr cs:[save_idt]

; at the end of the SMM routine the following code
; sequence will reload the entire CPU state
      mov     eax,cs:save_eax
      mov     ebx,cs:save_ebx
      mov     ecx,cs:save_ecx
      mov     edx,cs:save_edx
      mov     esi,cs:save_esi
      mov     edi,cs:save_edi
      mov     ebp,cs:save_ebp
      mov     esp,cs:save_esp
      rsdc    ds,cs:,save_ds
      rsdc    es,cs:,save_es
      rsdc    fs,cs:,save_fs
      rsdc    gs,cs:,save_gs
      rsdc    ss,cs:,save_ss
      rsldt   cs:,save_ldt
      rsts    cs:,save_tsr
      db      66h
      lgdt    fword ptr cs:[save_gdt]
      db      66h
      lidt    fword ptr cs:[save_idt]

; the data space so save the CPU state is in
; the Code Segment for this example
save_ds   dt     ?
save_es   dt     ?
save_fs   dt     ?
save_gs   dt     ?
save_ss   dt     ?
save_ldt  dt     ?
save_tsr  dt     ?
save_eax  dd     ?
save_ebx  dd     ?
save_ecx  dd     ?
save_edx  dd     ?
save_esi  dd     ?
save_edi  dd     ?
save_ebp  dd     ?
save_esp  dd     ?
save_gdt  df     ?
save_idt  df     ?
```

3.3.2 Maintaining Control Registers

CR0 is maintained in the SMM header. CR2 and CR3 should be saved if the SMM routine will be entering protected mode and enabling paging. Most SMM routines will not need to enable paging. However, if the CPU will be powered off, these registers should be saved.

3.3.3 Maintaining Debug Registers

DR7 is maintained in the SMM Header. Since DR7 is automatically initialized to the reset state on entry to SMM, the Global Disable bit (DR7 bit 13) will be cleared. This allows the SMM routine to access all of the Debug Registers. Returning from the SMM handler will reload DR7 with its previous value. In most cases, SMM routines will not make use of the Debug Registers and they will need to be saved only if the CPU needs to be powered down.

3.3.4 Maintaining Configuration Control Registers

The SMM routine should be written so that it maintains the Configuration Control Registers in the same state as they were initialized by the BIOS at power-up.

3.3.5 Maintaining FPU State

If power will be removed from the CPU or if the SMM routine will execute FPU instructions, then the FPU state should be maintained for the application running before SMM was entered. If the FPU state is to be saved and restored from within SMM, there are certain guidelines that must be followed to make SMM completely transparent to the application program.

The complete state of the FPU can be saved and restored with the FNSAVE and FNRSTOR instructions. FNSAVE is used instead of the FSAVE because FSAVE will wait for the FPU to check for existing error conditions before storing the FPU state. If there is a unmasked FPU exception condition pending, the FSAVE instruction will wait until the exception condition is serviced. To maintain transparency for the application program, the SMM routine should not service this exception. If the FPU state is restored with the FNRSTOR instruction before returning to normal mode, the application program can correctly service the exception. Any FPU instructions can be executed within SMM once the FPU state has been saved.

The information saved with the FSAVE instruction varies depending on the operating mode of the CPU. To save and restore all FPU information, the 32-bit protected mode version of the FPU save and restore instruction should be used. This can be accomplished by using the following code example:

```
; Save the FPU state
      mov    eax,CR0
      or     eax,00000001h
      mov    CR0,eax            ;set the PE bit in CR0
      jmp    $+2                ;clear the prefetch que
      db     66h                ;do 32bit version of fnsave
      fnsave [save_fpu]         ;saves fpu state to
                                ;the address DS:[save_fpu]
      mov    eax,CR0
      and    eax, 0FFFFFFFEh    ;clear PE bit in CR0
      mov    CR0,eax            ;return to real mode

;now the SMM routine can do any FPU instruction.
;Restore the FPU state before executing a RSM
      FNINIT                    ;initialize the FPU to a valid state
      mov    eax,CR0
      or     eax,00000001h
      mov    CR0,eax            ;set the PE bit in CR0
      jmp    $+2                ;clear the prefetch que
      db     66h                ;do 32bit version of fnsave
      frstor [save_fpu]         ;restore the FPU state
                                ;Some assemblers may require
                                ;use of the fnrstor instruction
      mov    eax,CR0
      and    eax, 0FFFFFFFEh    ;clear PE bit in CR0
      mov    CR0,eax            ;return to real mode
```

Be sure that all interrupts are disabled before using this method for entering protected mode. Any attempt to load a selector register while in protected mode will shutdown the CPU since no GDT is set up. Setting up a GDT and doing a long jump to enter protected mode will also work correctly.

3.4 Initializing the SMM Environment

After entering SMM and saving the CPU registers that will be used by the SMM routine, a few registers need to be initialized.

Segment registers need to be initialized if the CPU was operating in protected mode when the SMI interrupt occurred. Segment registers that will be used by the SMM routine should be loaded with known limits before they are used. The protected mode application could have set a segment limit to less than 64K. To avoid a protection error, all segment registers can be given limits of 4 GBytes. This can be done with the Cyrix RSDC instruction and will allow access to the full 4 GBytes of possible system memory without entering protected mode. Once the limits of a segment register are set, the base can be changed by use of the MOV instruction.

If necessary, an Interrupt Descriptor Table (IDT) should be set up in SMM memory before any interrupts or exceptions occur. The Descriptor Table Register can be loaded with an LIDT instruction to point to a small IDT in SMM memory that can handle the possible interrupts and exceptions that might occur while in the SMM routine.

A stack should always be set up in SMM memory so that stack operations done within SMM do not affect the system memory.

```
; SMM environment initialization example
include SMIMAC.INC             ; see Appendix A
      rsdc   ds,cs:,seg4G       ;DS is a 4GByte segment, base=0
      rsdc   es,cs:,seg4G       ;ES is a 4GByte segment, base=0
      rsdc   fs,cs:,seg4G       ;FS is a 4GByte segment, base=0
      rsdc   gs,cs:,seg4G       ;GS is a 4GByte segment, base=0
      rsdc   ss,cs:,seg4G       ;SS is a 4GByte segment, base=0
      lidt   cs:smm_idt         ;load IDT base and limit for
                                ;SMM's IDT
      mov    esp, smm_stack
      jmp    continue_smm_code
;
;descriptor of 4GByte data segment for use by rsdc
seg4G     dw        0ffffh       ; limit 4G
          dw        0            ; base = 0
          db        0            ; base = 0
          db        10010011B    ; data segment, DPL=0,P=1
          db        8fh          ; limit = 4G,
          db        0h           ; base = 0
          dw        0            ; segment register = 0
smm_idt   dw        smm_idt_limit
          dd        smm_idt_base
```

3.5 Accessing Main Memory Overlapped by SMM Memory

In SMM mode, there are instances where the program needs access to the system memory that is overlapping with SMM memory. This need for access this area of system memory most commonly occurs when the SMM routine is trying to save the entire memory image to disk before powering down the system. If using Cyrix SMM mode, access is made to main memory that overlaps SMM space by setting the MMAC bit in CCR1. The following code will enable and then disable MMAC.

```
; Set MMAC to access main memory
; this code is only valid for Cyrix SMM mode operations
MMAC = 1 shl 3
      mov    al, 0c1h           ;select CCR1
      out    22h, al
      in     al, 23h            ;get CCR1 current value
      mov    ah, al             ;save it
      mov    al, 0c1h           ;select CCR1 again
      out    22h, al
      mov    al, ah
      or     al, MMAC           ;set MMAC
      out    23h, al            ;write new value to CCR1
```

```
;Now all data memory access will use ADS#, Code fetches
;will continue to be done with SMADS# from SMM memory.
;
;Disable MMAC
      mov    al, 0c1h          ;select CCR1
      out    22h, al
      mov    al, ah            ;get old value of CCR1
      out    23h, al           ;and restore it
```

3.6 I/O Restart

Often when implementing a power management design, peripherals are required to be powered down by the system when not in use. When an I/O instruction is issued to a powered down device, the SMM routine is called to power up the peripheral and then reissue the I/O instruction. Cyrix CPUs make it easy to restart the I/O instruction that has generated an SMI interrupt.

The system will generate an SMI interrupt when an I/O bus cycle to a powered-down peripheral is detected. The SMM routine should interrogate the system hardware to find out if the SMI was caused by an I/O trap. By checking the SMM header information, the SMM routine can determine the type of I/O instruction that was trapped. If the I/O instruction has a REP prefix, the ECX register needs to be incremented before restarting the instruction. If the I/O trap was on a string I/O instruction, the ESI or EDI registers must be restored to their previous value before restarting the instruction.
The following code example shows how easy I/O restart is with the Cyrix CPU.

```
include SMIMAC.INC              ;see Appendix A
;Restart the interrupted instruction
      mov    eax,dword ptr cs:[SMI_CURRENTIP]
      mov    dword ptr cs:[SMI_NEXTIP],eax
      mov    al,byte ptr cs:[SMI_BITS]

;test for REP instruction
      bt     ax,2              ;rep instruction?
                               ;(result to Carry)
      adc    ecx,0             ;if so, increment ecx
      test   al,1 shl 1        ;test bit 1 to see
                               ;if an OUTS or INS
      jnz    out_instr
; A port read (INS or IN) instruction caused the
; chipset to generate an SMI instruction.
; Restore EDI from SMM header.
      mov    edi, dword ptr cs:[SMI_ESIEDI]
      jmp    common1

; A port write (OUTS or OUT) instruction caused the
; chipset to generate an SMI instruction.
; Restore ESI from SMM header.
out_instr:
      mov    esi, dword ptr cs:[SMI_ESIEDI]
common1:
```

3.7 I/O Port Shadowing and Emulation

Some system peripherals contain write-only ports. In a system that does power management, these peripherals need to be powered off and then reinitialized when their functions are needed later. The Cyrix SMM implementation makes it very easy to monitor the last value written to specific I/O ports. This process is known as shadowing. If the system can generate an SMI whenever specific I/O addresses get accessed, the SMM routine can, transparently to the system, monitor the port activity. The SMM header contains the address of the I/O write as well as the data. In addition, information is saved which indicates whether it is a byte, word or dword write. With this information, shadowing system write-only ports becomes trivial.

Some peripheral components contain registers that must be programmed in a specific order. If an SMI interrupt occurs while an application is accessing this type of peripheral, the SMI routine must be sure to reload the peripheral registers to the same stage before returning to normal mode. If the SMM routine needs to access such a peripheral, the previous normal-mode state must be restored. The previous accesses that were shadowed by previous SMM calls can be used to reload the peripheral registers back to the stage where the application was interrupted. The application can then continue where it left off accessing the peripheral.

In a similar way, the Cyrix SMM implementation allows the SMM routine to emulate the function of peripheral components in software.

3.8 Resume to HLT Instruction

To make an SMI interrupt truly transparent to the system, an SMI interrupt from a HLT instruction should return to the HLT instruction. There are known cases with DOS software where returning from an SMI handler to the instruction following the HLT will cause a system error. To determine if a HLT instruction was interrupted by the SMI, the H bit in the SMM header must be interrogated. If the H bit is set, the SMI interrupted a HLT instruction. To restart the HLT instruction simply decrement the NEXT_IP field in the SMM header.

The H bit is not available on a Cx486DX2/DX4. See Appendix B for a explanation on how to resume to a HLT instruction on a Cx486DX2/DX4.

```
;This is the start of specific code to check if the SMI
;occurred while in a HLT instruction.  If it did, then
;resume back to the HLT instruction when SMI is finished.

      include SMIMAC.INC                     ;see Appendix A


      mov         ax,cs:word ptr[SMI_BITS]     ;get H bit
      test        ax,0010h                     ;check if H=1
      je          not_hlt                      ;was not a HLT
      dec         cs:dword ptr[SMI_NEXTIP]     ;decrement NEXT_IP
not_hlt:
```

3.9 Exiting the SMI Handler

When the RSM instruction is executed at the end of the SMI handler, the EIP is loaded from the SMM header at the address (SMMbase + SMMsize - 14h) called NEXT_IP. This permits the instruction to be restarted if NEXT_IP was modified by the SMM program. The values of ECX, ESI, and EDI, prior to the execution of the instruction that was interrupted by SMI, can be restored from information in the header which pertains to the INx and OUTx instructions. See Section 3.6 for an example program to restart an I/O instruction. The only registers that are restored from the SMM header are CS, NEXT_IP, EFLAGS, CR0, and DR7. All other registers which were modified by the SMM program need to be restored before executing the RSM instruction.

3.10 Testing and Debugging SMM Code

An SMI routine can be debugged with standard debugging tools, such as DOS DEBUG, if the following requirements are met:

1. The debugger will only be able to set a code break point using INT 3 outside of the SMI handler. The debug control register DR7 is set to the reset value upon entry to the SMI handler. Therefore, any break conditions in DR0-3 will be disabled after entry to SMM. Debug registers can be used if they are set after entry to the SMI handler and if debug registers DR0-3 are saved.

2. The debugger must be running in real mode and the SMM routine must not enter protected mode. This insures that normal system interrupts, BIOS calls and the debugger will work correctly from SMM mode.

3. Before an INT 3 break point is executed, all segment registers should have their limits modified to 64K, or larger, within the SMM routine.

Appendix A

A. ASSEMBLER MACROS FOR CYRIX INSTRUCTIONS

The include file SMIMAC.INC provides a complex set of macros which generate SMM opcodes along with the appropriate mod/rm bytes. In order to function, the macros require that the labels which are accessed correspond to the specified segment. Thus segment overrides must be passed to the macro as an argument.

Do not specify a segment override if the default segment for an address is being used. If an address size override is used, a final argument of '1' must be passed to the macro as well. Address size overrides must be presented explicitly to prevent the assembler from generating them automatically and breaking the macros.

```
;SMM Instruction Macros - SMIMAC.INC
;Macros which generate mod/rm automatically

svdc    MACRO   segover,addr,reg,adover
        domac   segover,addr,reg,adover,78h
        ENDM
rsdc    MACRO   reg,segover,addr,adover
        domac   segover,addr,reg,adover,79h
        ENDM
svldt   MACRO   segover,addr,adover
        domac   segover,addr,es,adover,7ah
        ENDM
rsldt   MACRO   segover,addr,adover
        domac   segover,addr,es,adover,7bh
        ENDM
svts    MACRO   segover,addr,adover
        domac   segover,addr,es,adover,7ch
        ENDM
rsts    MACRO   segover,addr,adover
        domac   segover,addr,es,adover,7dh
        ENDM
rsm     MACRO
        db      0fh,0aah
        ENDM
smint   MACRO
        db      0fh,7eh
        ENDM
```

```
;Sub-Macro used by the above macro

domac   MACRO   segover,addr,reg,adover,op
        local   place1,place2,count
        count   = 0
        ifnb    <adover>
            count=count+1
        endif
        ifnb   <segover>
            count=count+1
        endif
        if      (count eq 0)
                nop             ;expanding the opcode one byte
        endif
        place1  = $
;pull off the proper prefix byte count
        mov     word ptr segover addr,reg
        org     place1+count
        mov     word ptr segover addr,reg
        place2  = $
;patch the opcode
        org     place1+(count*2)-1
        db      0Fh,op
        org     place2
ENDM

;Offset Definition for access into SMM space
SMI_SAVE STRUC
$ESIEDI         DD      ?
$IOWDATA        DD      ?
$IOWADDR        DW      ?
$IOWSIZE        DW      ?
$BITS           DD      ?
$CSSELL         DD      ?
$CSSELH         DD      ?
$CS             DW      ?
$RES1           DW      ?
$NEXTIP         DD      ?
$CURRENTIP      DD      ?
$CR0            DD      ?
$EFLAGS         DD      ?
$DR7            DD      ?
SMI_SAVE ENDS
```

```
SMI_ESIEDI      EQU  ($ESIEDI + SMMSIZE - SIZE SMI_SAVE)
SMI_IOWDATA     EQU  ($IOWDATA+ SMMSIZE - SIZE SMI_SAVE)
SMI_IOWADDR     EQU  ($IOWADDR+ SMMSIZE - SIZE SMI_SAVE)
SMI_IOWSIZE     EQU  ($IOWSIZE+ SMMSIZE - SIZE SMI_SAVE)
SMI_BITS        EQU  ($BITS   + SMMSIZE - SIZE SMI_SAVE)
SMI_CSSELL      EQU  ($CSSELL + SMMSIZE - SIZE SMI_SAVE)
SMI_CSSELH      EQU  ($CSSELH + SMMSIZE - SIZE SMI_SAVE)
SMI_CS          EQU  ($CS     + SMMSIZE - SIZE SMI_SAVE)
SMI_RES1        EQU  ($RES1   + SMMSIZE - SIZE SMI_SAVE)
SMI_NEXTIP      EQU  ($NEXTIP + SMMSIZE - SIZE SMI_SAVE)
SMI_CURRENTIP   EQU  ($CURRENTIP+ SMMSIZE -SIZE SMI_SAVE)
SMI_CR0         EQU  ($CR0    + SMMSIZE - SIZE SMI_SAVE)
SMI_EFLAGS      EQU  ($EFLAGS + SMMSIZE - SIZE SMI_SAVE)
SMI_DR7         EQU  ($DR7    + SMMSIZE - SIZE SMI_SAVE)
```

SMM Instruction macro example: TEST.ASM

```
                                    .MODEL  SMALL
                                    .386
                                    ;SMM Macro Examples


                                    include smimac.inc

0000                            .DATA
0000  0A*(??)                   there   db      10 dup (?)
000A                            .CODE
0000  2E 0F 78 1E 004E                 svdc    cs:,hello,ds
0006  2E 0F 79 1E 004E                 rsdc    ds,cs:,hello
000C  2E 0F 79 2E 004E                 rsdc    gs,cs:,hello
0012  2E 67 2E 0F 78 9C 58 0000004E svdc    cs:,[eax+ebx*2+hello],1
001D  67| 0F 78 23                     svdc    ,[ebx],fs,1

0021  0F 78 2E 0000                    svdc    ,there,gs
0026  2E 0F 7A 06 004E                 svldt   cs:,hello
002C  2E 0F 7B 06 004E                 rsldt   cs:,hello

0032  2E 0F 7D 06 004E                 rsts    cs:,hello
0038  2E 67 2E 0F 7C 84 58 0000004E svts    cs:,[eax+ebx*2+hello],1
0043  67| 0F 7A 03                     svldt   ,[ebx],1
0047  0F 7C 06 0000                    svts    ,there
004C  0F AA                            rsm

004E  0A*(??)                          hello   db   10 dup (?)
end
```

B. DX2/DX4 Resume to Halt

The Cx486DX2/DX4 does not support the H bit in the SMM header. To make an SMI interrupt truly transparent to the system, an SMI interrupt from a HLT instruction should return to the HLT instruction. There are known cases with DOS software where returning from an SMI handler to the instruction following the HLT will cause a system error. To determine if a HLT instruction was interrupted by the SMI, the opcode from memory needs to be interrogated. This code example describes how to determine if the current instruction is a HLT and how to restart it.

```
;This is the start of specific code to check if the SMI
;occurred while in a HLT instruction. If it did, then
;return back to the HLT instruction when SMI is finished.

      rsdc   fs,cs:,[seg4G]     ;FS is base=0 limit=4G data
                                ;segment to be used to check if
                                ;HLT instruction was executing

;on a Cyrix part, if the SMI occurred while in a HLT
;instruction, the CURRENT IP and the NEXT IP will both
;point to the instruction following the HLT.
      mov    eax,cs:dword ptr[SMI_CURRENTIP]
      cmp    eax,cs:dword ptr[SMI_NEXTIP]
      jne    not_hlt            ;can't be a HLT but could be
                                ;a LOOP or REP
;load EAX with CS base from the SMM header
      mov    ax,cs:word ptr [SMI_CSSELH+2]
      mov    al,cs:byte ptr [SMI_CSSELH]
      shl    eax,10h
      mov    ax,cs:word ptr[SMI_CSSELL+2]
;calculate linear address
      add    eax,cs:dword ptr [SMI_CURRENTIP]
      dec    eax                ;decrement to HLT instruction
      mov    edx,eax            ;save lin addr in edx
      mov   eax,cs:dword ptr [SMI_CR0] ;check if paging on
      test  eax,80000000h
      je    no_paging           ;if no paging then linear
                                ;address = physical address
;set MMAC to get access to Main memory
      mov   al,0c1h
      out   22h,al
      in    al,23h
      mov   cl,al               ;save old CCR1 value in cl
      mov   al,0c1h
      out   22h,al
      mov   al,cl
      or    al,08h              ;set MMAC bit in CCR1
      mov   al,0c1h
      out   23h,al
      mov   eax,CR3             ;get Page Directory Base Reg
      and   eax,0fffff000h
      mov   ebx,edx             ;linear address
      shr   ebx,22              ;get 10 byte Directory Entry
;read Directory Table
      mov   eax,dword ptr fs:[eax+ebx*4]
      and   eax,0fffff000h
      mov   ebx,edx             ;linear address
      shr   ebx,12
      and   ebx,03ffh           ;get 10 byte Page Table Entry
      mov   eax,dword ptr fs:[eax+ebx*4]
      and   eax,0fffff000h
      mov   ebx,edx             ;linear address
      and   ebx,0fffh           ;get 12 byte offset into page
;Get the physical address of the instruction before the
;Current IP.  Save in BL.
      mov   bl,byte ptr fs:[eax+ebx]
      mov   al,0c1h             ;set MMAC back to normal
      out   22h,al
      mov   al,cl
      out   23h,al              ;MMAC = 0
      jmp   got_inst

;If paging is not enabled then checking for the HLT
;instruction is easy since the linear address equals
;the physical address.

no_paging:
      mov   al,0c1h             ;set MMAC
      out   22h,al
      in    al,23h
      mov   ah,al
      mov   al,0c1h
      out   22h,al
      mov   al,ah
      or    al,08h
      out   23h,al
;get instruction interrupted by SMI
      mov   bl,byte ptr fs:[edx]
      mov   al,0c1h             ;store it in BL
      out   22h,al
      mov   al,ah
      out   23h,al              ;set MMAC back to normal

got_inst:
      cmp   bl,0f4h             ;was it a HLT instruction?
      jne   not_hlt             ;if not a F4 then not a HLT
                                ;set up SMM header to return
                                ;to the HLT instruction
      dec   cs:dword ptr [SMI_NEXTIP]

not_hlt:
      jmp    continue_SMI_routine

; data within the SMM Space Code Segment
seg4G dw      0ffffh            ;limit 15-0
      dw      0                 ;base
      db      0                 ;base
      db      10010011B         ;data segment, DPL=0, present
      db      8Fh               ;high limit =f, Gran =4K, 16 bit
      db      0                 ;base
      dw      0
```

Cyrix®
Advancing the Standards

C

C. Differences in Cyrix Processors

Table C-1 lists the major differences between the Cx486DX2/DX4, 5x86 and 6x86 CPUs as related to System Management Mode.

Table C-1. Differences between Cyrix CPUs

Feature	Cx486DX2/DX4	5x86	6x86
SMAC CCR1 - bit 2	Valid only. if SMM_MODE=0.	Valid only if SMM_MODE=0.	Available
MMAC CCR1 - bit 3	Valid only. if SMM_MODE=0.	Valid only. if SMM_MODE=0.	Not available
SM3 CCR1 - bit 7	Not available register index CDh, CEh and CFh are always defined as SMAR.	Not available, register index CDh, CEh and CFh are always defined as SMAR.	Must be set to define register index CDh CEh and CFh as SMAR.
SMIACT CCR3 - bit3	Available on revisions with DIR1 >= 30h. Prior revisions only support Cyrix SMM Mode.	Available	Always in SL SMM mode.
SMAR SIZE field	If = Fh, SMAR size set to 4 KBytes	If = Fh, SMAR size set to 4K Bytes	If = Fh, SMAR size set to 4 GBytes

SMI# acknowledged when:	CPL=0 &
	USE_SMI=1 &
	(SMAR size > 0) &
	SMAC=0 &
	(in normal mode)
		CPL=0 &
		USE_SMI=1 &
		(SMAR size > 0) &
		SMAC=0 &
		(in normal mode)
			CPL=0 &
			USE_SMI=1 &
			(ARR3 size > 0) &
			SM3=1 &
			SMAC=0 &
			(in normal mode)

SMINT instruction is valid when:	CPL=0 & USE_SMI=1 &
	(SMAR size > 0) &
	SMAC=1 &
	SMM_Mode=0
		CPL=0 & USE_SMI=1 &
		(SMAR size > 0) &
		SMAC=1 &
		SMM_Mode=0
			CPL=0 & USE_SMI=1 &
			(ARR3 size > 0) &
			SM3=1 &
			SMAC=1

Cyrix Specific SMM instructions are valid when:	CPL=0 & USE_SMI=1 & (SMAR size > 0) & (SMAC=1 or in SMM mode)	CPL=0 & USE_SMI=1 & (SMAR size > 0) & (SMAC=1 or in SMM mode)	CPL=0 & USE_SMI=1 & (ARR3 size > 0) & SM3=1 & (SMAC=1 or in SMM mode)

Table C-1. Differences between Cyrix CPUs (Continued)

Feature	Cx486DX2/DX4	5x86	6x86
H bit in SMM header	Not available, Reserved See Appendix B for details for resuming to a HLT instruction.	Valid	Valid
I/O trap information	I/O Data Size, I/O Address and I/O Data only valid for I/O writes trapped by an SMI.	I/O Data Size, I/O Address and I/O Data valid for both I/O reads and writes trapped by an SMI.	I/O Data Size, I/O Address and I/O Data valid for both I/O reads and writes trapped by an SMI.
CS limit on entry to SMM	64 KByte limit	4 GByte limit	4 GByte limit
CR0 value on entry to SMM	0000 0010h except EM bit is not cleared on entry. The SMM routine should clear EM before executing any FPU instructions.	6000 0010h if LOCK_NW=1 then NW is not changed	6000 0010h if LOCK_NW=1 then NW is not changed

Cyrix Worldwide Offices

United States
Corporate Office
Richardson, Texas
Tel: (214) 968-8388
Fax: (214) 699-9857

Tech Support and Sales: (800) 462-9749
Internet: tech_support@cyrix.com
BBS: (214) 968-8610 (up to 28.8K baud)

See us on the Internet Worldwide Web:
http://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®
Advancing the Standards

Cyrix Corporation
P.O. Box 850118
Richardson, TX 75085-0118
Tel: (214) 968-8388
Fax: (214) 699-9857

94211-01 March 1996

Printed in the USA on recycled paper.