Cyrix III Processor DataBook

Socket 370 Compatible x86 CPU Featuring
MMX™ and 3DNow!™ Technology

Trademark Acknowledgments:

Cyrix is a registered trademark of Via Cyrix Corporation.
Cyrix III is a trademark of Via-Cyrix Corporation. MMX is a trademark of Intel Corporation.
All other brand or product names are trademarks of their respective companies.

Via-Cyrix Corporation
2703 North Central Expressway
Richardson, Texas 75080-2010
United States of America

REVISION HISTORY

Date	Version	Revision
1/25/00	1.0	Final Specs updated for production
3/22/99	0.53	Changed name from MXs to Cyrix III processor.
3/17/99	0.52	Pages 1-9, 1-10: Both L1 and L2 caches are unified. Page 1-11 Remove paragraph concerning Scratch Pad Cache Memory Page 2-36 and similar pages Replace “Don’t care” with xxxxb. Page 2-56 Added Clock Ratio Table for BIOS Core/Bus Frequency Ratios Added CPUPRES# signal. Changed X32 pin to GND Changed Z32 pin to Vcc Changed name VDD-2.5 to VCC_2.5 Changed AD32 pin to Vcc Added IERR# signal Added BRO# Modified Figure 5.3 Voltage Connections Changed REF7 to VREF7 in pin diagrams.
3/16/99	0.51	Corrected Pages 5-1 and 5-2 Pin Assignment Diagrams Corrected Pages 5-3 and 5-4 Pin Signal List Added power diagram Page 5-6 Figure 5-3.
3/11/99	0.5	Page 2-33 Updated and completed CPU Configuration Register Table Page 2-43 Added question marks --Does CCR7 exist? Page 2-48 Updated RCRn bit 0 RCD/RCE Page 2-53 Added Clock Ratio Table for DIR3 TYPE Field
3/9/99	0.4	Bullet Page:Added Programmable Clock/Bus Ratio, new ratio Page 3-1 Made several changes to Figure 3-1. Page 3-2 Removed subname explanation. Page 3-2 Redefined ADS#. Page 3-3 LOCK# is now I/O. Page 3-5 Many changes to Table 3-2, non-supported signals. Page 3-9 Rewrote second paragraph right column. Page 3-10 Omitted table 3-6 as all signals are disconnected during RESET#. Page 3-13 Placed Error Phase before Snoop Phase. Page 3-14 Rewrote INTR explanation. Now there is one interrupt acknowledge bus cycle. Page 3-15 Next to last paragraph. Removed last sentence concerning FERR#. Page 4-2 Made many changes to Table 4-1 Pull-Up Resistors. Page 4-3 Table 4-2 Note 3. Removed the word “APIC”. Page 4-4 Table 4-3, Recommended Operating Conditions for CMOS Signals. Removed V_{CCIO} row. Page 4-5 Moved BSEL signals from GTL I/O to CMOS Input row. Removed LINT signals. Page 4-7 and 4-8 Added 433, 450, 500 MHz frequencies.
3/3/99	0.3	Page 1-9, reworded right-top paragraph concerning exclusive cache. Corrected BSEL0 and BSEL1 typos on page 3-1 and 4-5. Added 133 MHz bus on page 3-9. Corrected note to Figure 4-14 on page 4-12. Updated Figure 4-15 on page 4-11.
3/1/99	0.2	Typos. Added 2.5x to Table 3-3. Updated thermal information.
2/18/99	0.1	Initial Version C:\documentation\joshua\CyrixIII_0.fm

Cyrix III PROCESSOR

Socket 370 Compatible CPU

MMX™ and 3DNow!™ Technology

Cyrix Processors

Introduction

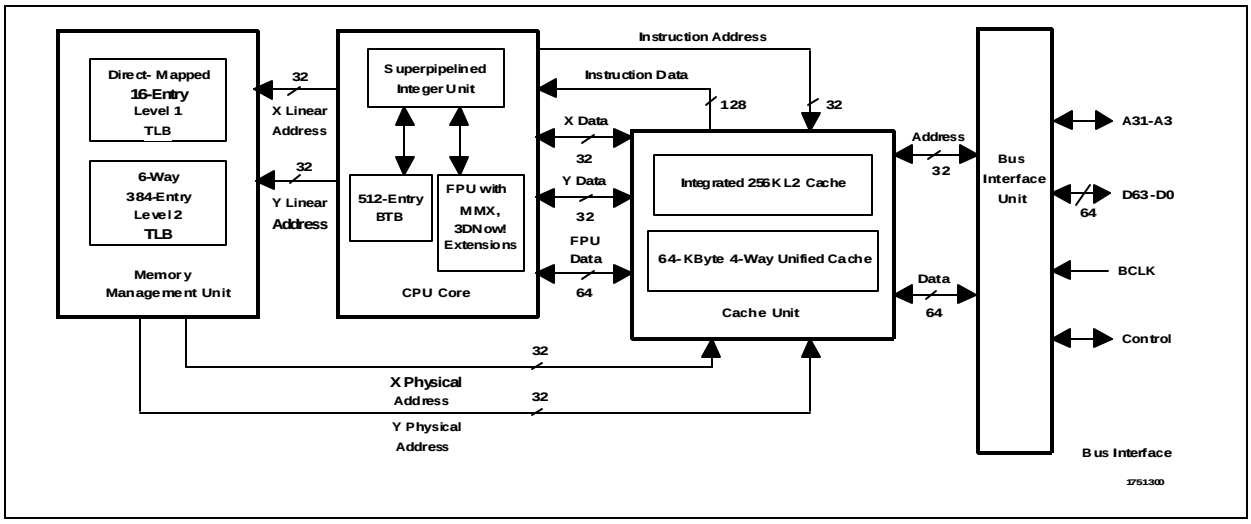
- Performance Features
  - Performance Rating: PR400, PR450, PR500 and higher
  - Integrated 8-way 256 KByte L2 Cache
  - 64K 4-Way Unified Write-Back L1 Cache
  - Branch Prediction with a 512-entry BTB
  - Enhanced Memory Management Unit 2 Level TLB (16 Entry L1, 384 Entry L2)
  - Scratchpad RAM in Unified Cache
  - Optimized pipelining for both 32- and 16-Bit Code
  - High Performance dual pipeline 80-Bit FPU
  - Supports bus speeds of 66, 100, and 133 Mhz

- Other Features
  - Leverages Existing Socket 370 Infrastructure
  - Compatible with MMX™ and 3D Now!™ Technology
  - Runs Windows® 98, Windows 3.x, Windows NT, DOS, UNIX®, OS/2®, and all other x86 operating systems.
  - 2.2 V Core
  - Flexible Core/Bus Clock Ratios: 2.5x, 3x, 3.5x, 4.0x, 4.5x, 5.0x, 5.5x, 6.0x, 6.5x, 7.0x, 7.5x
  - BIOS Programmable Core/Bus Clock Ratio

The Cyrix III processor offers significant performance enhancements over previous generation processors in a Socket 370 compatible package. The Cyrix III includes a 64 KByte L1 cache, an integrated 256 KByte L2 cache, and has frequency scalability to 400 MHz and beyond. It is compatible with MMX and 3DNow! Technology for superior graphics performance. A new dual pipeline FPU/MMX Unit delivers superior floating point performance. The Cyrix III delivers high 32-bit and 16-bit performance while running Windows 98, 95 and 3.X, Windows NT, OS/2, DOS, UNIX, and all other x86 operating systems and applications.

The Cyrix III processor achieves top performance through the use of two optimize superpipelined execution units, two integer units, and a dual issue FPU/MMX/3DNow! unit that can execute instructions from both execution units simultaneously. The 64 K L1 cache and 256 K integrated L2 cache employ write-back technologies to make access to the code and data as fast as possible to avoid pipeline stalls. The cache supports caching SMI code and data, and can be used as scratchpad RAM by the processor.

The superpipelined architecture reduces timing constraints and increases frequency scalability. Advanced architectural technologies include register renaming, out of order completion, data dependency removal, branch prediction, and speculative execution. The pipelining and superscaling are designed to remove data dependencies and resolve conflicts to allow for a high number of instruction executions per clock cycle. This promotes the highest performance for both 32 bit and 16-bit applications.



Cyrix Processors

Table of Contents

Table of Contents

Cyrix Processors

Product Overview

1 ARCHITECTURE OVERVIEW

The Cyrix III processor is a 64-bit, x86 instruction set compatible processor that provides high-performance in a Celeron™ compatible PGA 370 socket. The Cyrix III processor offers an enhanced super-scalar core, and a new pipeline, dual-issue MMX™ and 3DNOW!™ -compatible floating point unit (FPU). The Cyrix III processor can process 57 new multimedia instructions compatible with MMX™ technology.

The processor contains a 64K L1 cache and a 256K L2 cache. It operates at a higher frequency, contains an enlarged cache, a two-level TLB, and an improved branch target cache.

The Cyrix III processor core is an enhanced version of a proven design that offers competitive CPU performance. It has integer and floating point execution units that are based on sixth-generation technology. The integer core contains a dual-issue, seven-stage execution pipeline and offers advanced features such as operand forwarding, branch target buffers, and extensive write buffering. The FPU has been redesigned to provide additional buffering, reduced latency, and improved throughput up to 1 GFLOP (peak). The dual issue FPU can allow two MMX or floating point instructions to execute simultaneously. A 64KB write-back L1 cache is accessed in a unique fashion that eliminates pipeline stalls for fetch operands that hit in the cache.

Through the use of unique architectural features, the Cyrix III processor eliminates many data dependencies and resource conflicts, resulting in optimal performance for both 16-bit and 32-bit x86 software.

To provide support for multimedia operations, the cache can be turned into a scratchpad RAM memory on a line by line basis. The cache area set aside as scratchpad memory acts as a private memory for the CPU and does not participate in cache operations.

The on-chip FPU has been enhanced to process MMX™ and 3DNow! instructions as well as the floating point instructions. Both types of instructions execute in parallel with integer instruction processing. To facilitate FPU operations, the FPU features a 64-bit data interface, a four-deep instruction queue and a six-deep store queue.

For mobile systems and other power sensitive applications, the Cyrix III processor incorporates low power suspend mode, stop clock capability, and system management mode (SMM).

Cyrix Processors

1.1 Processor Differences

Tables 1 describe the major differences between the M II and Cyrix III processors.

Table 1-1. Cyrix III Processor vs. M II Processor

Feature	Cyrix III Processor	M II Processor
Package/pinout	Socket 370 SPGA	Socket 7
Supply voltage	Core voltage = 2.2v i/o reference voltage = 1.0v	Core voltage = 2.9v I/O voltage = 3.3v
CPU primary cache (L1)	64 KB write-back L1 Cache	64 KB write-back Cache
Support for secondary cache (L2)	Internal L2 Cache, 256KBs	Yes (512K external typical)
MMX™ Instruction Set	Yes	Yes
3DNOW!™ Instruction Set	Yes	No
Floating point unit	Dual-issue from Integer Unit	Single-issue from Integer Unit
4MB paging	Yes	No
Virtual Mode Extensions	Yes	No

1.2 Celeron™ Compatibility

The Cyrix III processor is design to be compatible with motherboards created for the Intel® Celeron processor with a socket 370 footprint. However some electrical signaling differs so that the Cyrix III can provide features not supported by the Celeron. In particular, the Cyrix III support two unique pins, the VID[4] pin AK36 used to signal 2.2 volt operation and the BSEL1 pin AK30 used with BSEL0 pin AJ33 to signal system bus frequency.

Conversely, a few minor Celeron signals are not supported by the Cyrix III processor and include: breakpoint signals (BP[3:2] and BPM[1:0]#; internal error signal (IERR#); probe signals (PRDY#, PREQ#); and thermal trip signal (THERMTRIP#).

Refer to chapter 3 of this manual for more details on Cyrix III signal descriptions. For mother-board design considerations and more details concerning Celeron compatibility refer to the *Cyrix III Board Design and AC/DC Specifications* Application Note 120.

1.3 Major Functional Blocks

The Cyrix III processor consists of four major functional blocks, as shown in the overall block diagram on the first page of this manual:

- Memory Management Unit
- CPU Core
- Cache Unit
- Bus Interface Unit

The CPU contains the superpipelined integer unit, the BTB (Branch Target Buffer) unit and the FPU (Floating Point Unit).

The BIU (Bus Interface Unit) provides the interface between the external system board and the processor's internal execution units. During a memory cycle, a memory location is selected through the address lines (A[31-3]#). Data is passed from or to memory through the data lines (D[63-0]#).

Each instruction is read into 256-Byte Instruction Line Cache. The Cache Unit stores the most recently used data and instructions to allow fast access to the information by the Integer Unit and FPU.

The CPU core requests instructions from the Cache Unit. The received integer instructions are decoded by either the X or Y processing pipelines within the superpipelined integer unit. If the instruction is a MMX or FPU instruction it is passed to the floating point unit for processing.

Data is fetched from the 64-KB unified cache as required. If the data is not in the cache it is accessed via the bus interface unit from main memory.

The Memory Management Unit calculates physical addresses including addresses based on paging.

Physical addresses are calculated by the Memory Management Unit and passed to the Cache Unit and the Bus Interface Unit (BIU).

1.4 Integer Unit

The Integer Unit (Figure 1-1) provides parallel instruction execution using two seven-stage integer pipelines. Each of the two pipelines, XandY, can process several instructions simultaneously.

The Integer Unit consists of the following pipeline stages:

- Instruction Fetch (IF)
- Instruction Decode 1 (ID1)
- Instruction Decode 2 (ID2)
- Address Calculation 1 (AC1)
- Address Calculation 2 (AC2)
- Execute (EX)
- Write-Back (WB)

The instruction decode and address calculation functions are both divided into superpipelined stages.

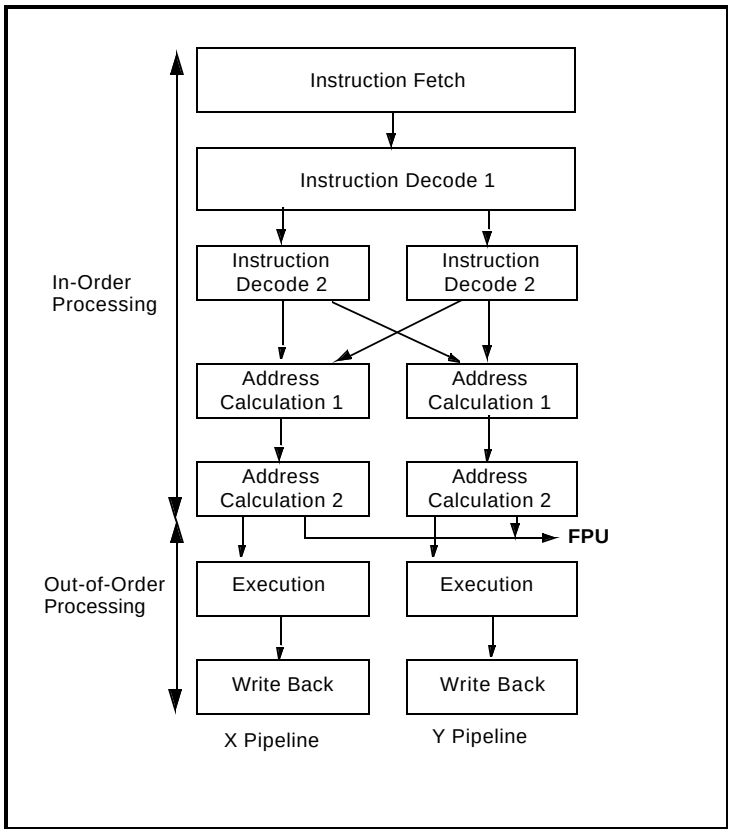


Figure 1-1. Integer Unit

1.4.1 Pipeline Stages

The Instruction Fetch (IF) stage, shared by both the X and Y pipelines, fetches 16 bytes of code from the cache unit in a single clock cycle. Within this section, the code stream is checked for any branch instructions that could affect normal program sequencing.

If an unconditional or conditional branch is detected, branch prediction logic within the IF stage generates a predicted target address for the instruction. The IF stage then begins fetching instructions at the predicted address.

The superpipelined Instruction Decode function contains the ID1 and ID2 stages. ID1, shared by both pipelines, evaluates the code stream provided by the IF stage and determines the number of bytes in each instruction. Up to two instructions per clock are delivered to the ID2 stages, one in each pipeline.

The ID2 stages decode instructions and send the decoded instructions to either the X or Y pipeline for execution. The particular pipeline is chosen, based on which instructions are already in each pipeline and how fast they are expected to flow through the remaining stages.

The Address Calculation function contains two stages, AC1 and AC2. If the instruction refers to a memory operand, the AC1 calculates a linear memory address for the instruction.

The AC2 stage performs any required memory management functions, cache accesses, and register file accesses. If a floating point instruction is detected by AC2, the instruction is sent to the FPU for processing.

The Execute (EX) stage executes instructions using the operands provided by the address calculation stage.

The Write-Back (WB) stage is the last IU stage. The WB stage stores execution results either to a register file within the IU or to a write buffer in the cache control unit.

1.4.2 Out-of-Order Processing

If an instruction executes faster than the previous instruction in the other pipeline, the instructions may complete out of order. All instructions are processed in order, up to the EX stage. While in the EX and WB stages, instructions may be completed out of order.

If there is a data dependency between two instructions, the necessary hardware interlocks are enforced to ensure correct program execution. Even though instructions may complete out of order, exceptions and writes resulting from the instructions are always issued in program order.

1.4.3 Pipeline Selection

In most cases, instructions are processed in either pipeline and without pairing constraints on the instructions. However, certain instructions are processed only in the X pipeline:

- Branch instructions
- Floating point instructions
- Exclusive instructions

Branch and floating point instructions may be paired with a second instruction in the Y pipeline.

Exclusive Instructions cannot be paired with instructions in the Y pipeline. These instructions typically require multiple memory accesses. Although exclusive instructions may not be paired, hardware from both pipelines is used to accelerate instruction completion. Listed below are the Cyrix III CPU exclusive instruction types:

- Protected mode segment loads
- Special register accesses (Control, Debug, and Test Registers)
- String instructions
- Multiply and divide
- I/O port accesses
- Push all (PUSHA) and pop all (POPA)
- Intersegment jumps, calls, and returns

1.4.4 Data Dependency Solutions

When two instructions that are executing in parallel require access to the same data or register, one of the following types of data dependencies may occur:

- Read-After-Write (RAW)
- Write-After-Read (WAR)
- Write-After-Write (WAW)

Data dependencies typically force serialized execution of instructions. However, the Cyrix III CPU implements three mechanisms that allow parallel execution of instructions containing data dependencies:

- Register Renaming
- Data Forwarding
- Data Bypassing

The following sections provide detailed examples of these mechanisms.

1.4.4.1 Register Renaming

The Cyrix III CPU contains 32 physical general purpose registers. Each of the 32 registers in the register file can be temporarily assigned as one of the general purpose registers defined by the x86 architecture (EAX, EBX, ECX, EDX, ESI, EDI, EBP, and ESP). For each register write operation a new physical register is selected to allow previous data to be retained temporarily. Register renaming effectively removes all WAW and WAR dependencies. The programmer does not have to consider register renaming as register renaming is completely transparent to both the operating system and application software.

1.4.4.2 Data Forwarding

Register renaming alone cannot remove RAW dependencies. The Cyrix III CPU uses two types of data forwarding in conjunction with register renaming to eliminate RAW dependencies:

- Operand Forwarding
- Result Forwarding

Operand forwarding takes place when the first in a pair of instructions performs a move from register or memory, and the data that is read by the first instruction is required by the second instruction. The Cyrix III CPU performs the read operation and makes the data read available to both instructions simultaneously.

Result forwarding takes place when the first in a pair of instructions performs an operation (such as an ADD) and the result is required by the second instruction to perform a move to a register or memory. The Cyrix III CPU performs the required operation and stores the results of the operation to the destination of both instructions simultaneously.

Operand forwarding can only occur if the first instruction does not modify its source data. In other words, the instruction is a move type instruction (for example, MOV, POP, LEA). Operand forwarding occurs for both register and memory operands. The size of the first instruction destination and the second instruction source must match.

1.5 Data Bypassing

In addition to register renaming and data forwarding, the Cyrix III CPU implements a third data dependency-resolution technique called data bypassing. Data bypassing reduces the performance penalty of those memory data RAW dependencies that cannot be eliminated by data forwarding.

Data bypassing is implemented when the first in a pair of instructions writes to memory and the second instruction reads the same data from memory. The Cyrix III CPU retains the data from the first instruction and passes it to the second instruction, thereby eliminating a memory read cycle. Data bypassing only occurs for cacheable memory locations.

1.5.1 Branch Control

Branch instructions occur on average every four to six instructions in x86-compatible programs. When the normal sequential flow of a program changes due to a branch instruction, the pipeline stages may stall while waiting for the CPU to calculate, retrieve, and decode the new instruction stream. The Cyrix III CPU minimizes the performance degradation and latency of branch instructions through the use of branch prediction and speculative execution.

1.5.1.1 Branch Prediction

The Cyrix III CPU uses a 512-entry, 4-way set associative Branch Target Buffer (BTB) to store branch target addresses. The Cyrix III CPU has 1024-entry branch history table. During the fetch stage, the instruction stream is checked for the presence of branch instructions. If an unconditional branch instruction is encountered, the Cyrix III CPU accesses the BTB to check for the branch instruction's target address. If the branch instruction's target address is found in the BTB, the Cyrix III CPU begins fetching at the target address specified by the BTB.

In case of conditional branches, the BTB also provides history information to indicate whether the branch is more likely to be taken or not taken. If the conditional branch instruction is found in the BTB, the Cyrix III CPU begins fetching instructions at the predicted target address. If the conditional branch misses in the BTB, the Cyrix III CPU predicts that the branch will not be taken, and instruction fetching continues with the next sequential instruction. The decision to fetch the taken or not taken target address is based on a four-state branch prediction algorithm.

Once fetched, a conditional branch instruction is first decoded and then dispatched to the X pipeline only. The conditional branch instruction proceeds through the X pipeline and is then resolved in either the EX stage or the WB stage. The conditional branch is resolved in the EX stage, if the instruction responsible for setting the condition codes is completed prior to the execution of the branch. If the instruction that sets the condition codes is executed in parallel with the branch, the conditional branch instruction is resolved in the WB stage.

Correctly predicted branch instructions execute in a single core clock. If resolution of a branch indicates that a misprediction has occurred, the Cyrix III CPU flushes the pipeline and starts fetching from the correct target address. The Cyrix III CPU prefetches both the

predicted and the non-predicted path for each conditional branch, thereby eliminating the cache access cycle on a misprediction. If the branch is resolved in the EX stage, the resulting misprediction latency is four cycles. If the branch is resolved in the WB stage, the latency is five cycles.

Since the target address of return (RET) instructions is dynamic rather than static, the Cyrix III CPU caches target addresses for RET instructions in an eight-entry return stack rather than in the BTB. The return address is pushed on the return stack during a CALL instruction and popped during the corresponding RET instruction.

1.5.1.2 Speculative Execution

The Cyrix III CPU is capable of speculative execution following a floating point instruction or predicted branch. Speculative execution allows the pipelines to continuously execute instructions following a branch without stalling the pipelines waiting for branch resolution. The same mechanism is used to execute floating point instructions in parallel with integer instructions.

The Cyrix III CPU is capable of up to four levels of speculation (i.e., combinations of four conditional branches and floating point operations). After generating the fetch address using branch prediction, the CPU checkpoints the machine state (registers, flags, and processor environment), increments the speculation level counter, and begins operating on the predicted instruction stream.

Once the branch instruction is resolved, the CPU decreases the speculation level. For a correctly predicted branch, the status of the checkpointed resources is cleared. For a branch misprediction, the Cyrix III processor generates the correct fetch address and uses the checkpointed values to restore the machine state in a single clock.

In order to maintain compatibility, writes that result from speculatively executed instructions are not permitted to update the cache or external memory until the appropriate branch is resolved. Speculative execution continues until one of the following conditions occurs:

1) A branch or floating point operation is decoded and the speculation level is already at four.

2) An exception or a fault occurs.

3) The write buffers are full.

4) An attempt is made to modify a non-checkpointed resource (i.e., segment registers, system flags).

1.6 Cache Units

The Cyrix III CPU employs two caches, the 64KB L1 Cache and the Exclusive L2 Cache (Figure 1-2, Page 1-10). The main cache is a 4-way set-associative 64-KB unified cache. The unified cache provides a higher hit rate than using equal-sized separate data and instruction caches. While in Cyrix SMM mode both SMM code and data are cacheable.

To avoid data conflicts, both caches are exclusive, that is data can be stored in either cache but not both at the same time.

1.6.1 64-KB L1 Cache

The 64-KB unified write-back cache functions as the primary cache. Configured as a four-way set-associative cache, the cache stores up to 64Kilobytes of code and data in 2048 lines. The cache is dual-ported and allows any two of the following operations to occur in parallel:

- Code fetch
- Data read (X pipe, Y pipeline or FPU)
- Data write (X pipe, Y pipeline or FPU)

The unified cache uses a pseudo-LRU replacement algorithm and can be configured to allocate new lines on read misses only or on read and write misses.

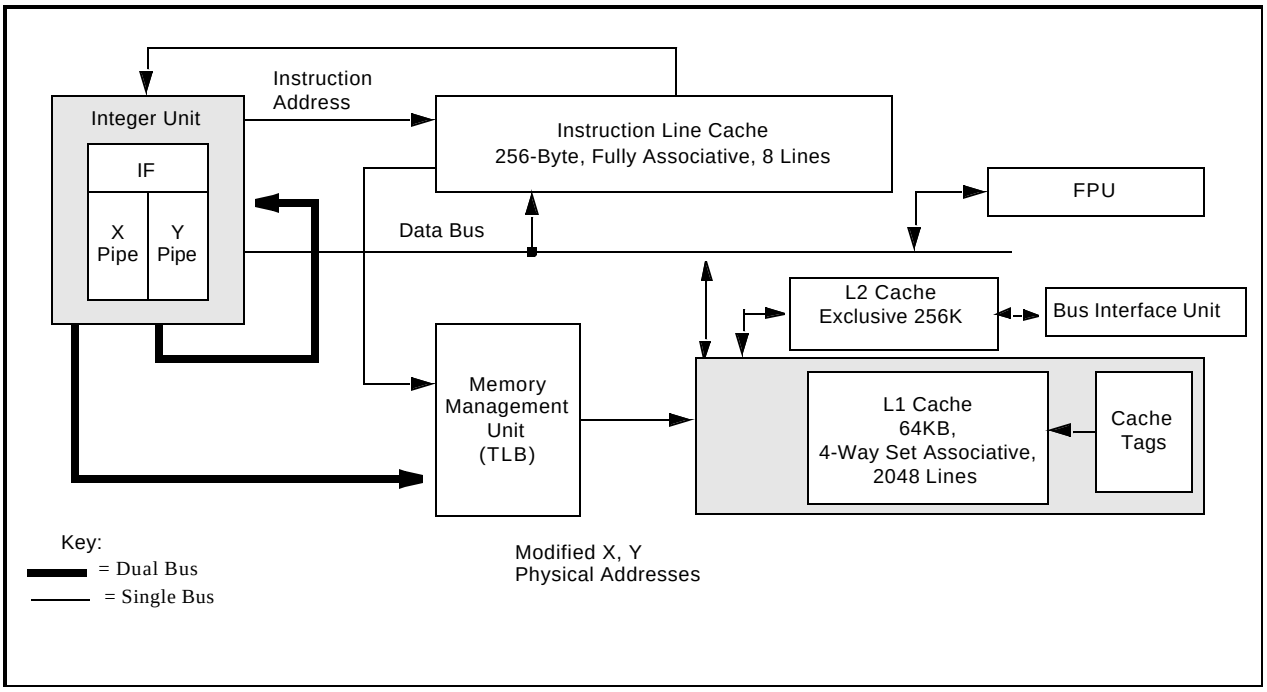


Figure 1-2. Cache Unit Operations

1.6.2 Exclusive L2 Cache

The exclusive 256 KB L2 cache serves as a unified secondary cache. This L2 cache is filled through L1 cache eviction. Fetches from the integer unit that do not hit in the L1 cache will access the L2 cache. The L2 can act as a victim cache, saving cache lines released by the L1 cache. The L2 cache is thus referred to as an exclusive, or victim cache, since it only contains data that is not found in the L1 cache. The L2 cache is 8-way set associative.

The total cache of the Cyrix III CPU can be up to 320k since the exclusive L2 architecture ensures that no data will be in both the L1 and L2. This also eliminates the need for an L1 to L2 writeback cycle, thus improving performance.

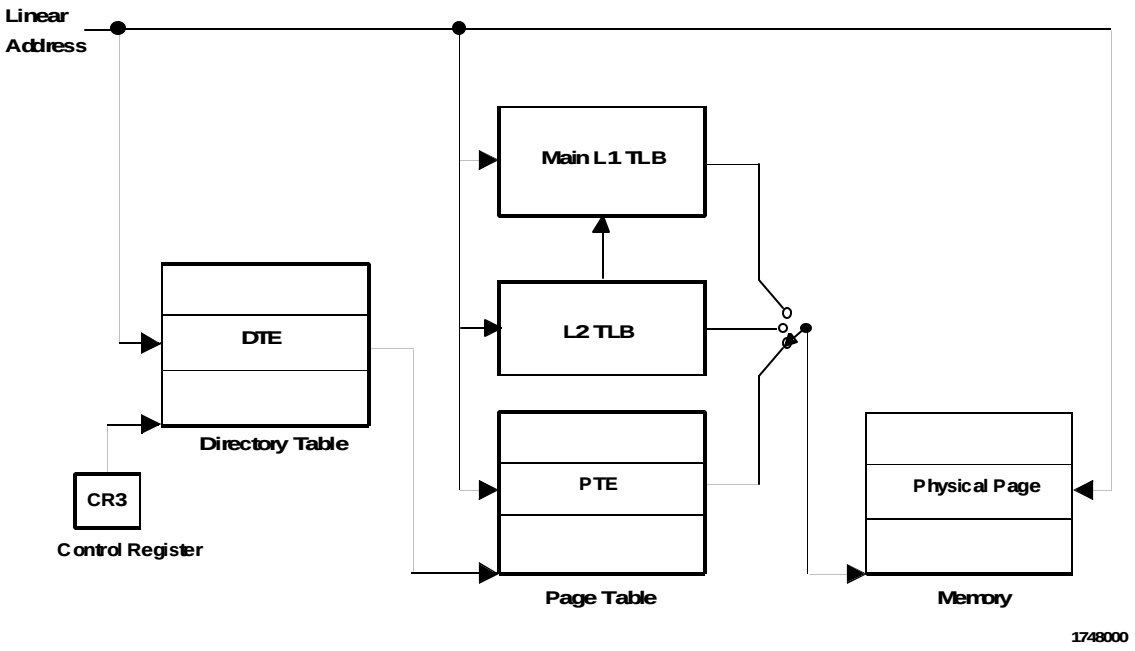
The L2 cache bus operates at the same frequency as the cpu core, delivering cache data at very high speed to the execution units.

1.7 Memory Management Unit

The Memory Management Unit (MMU), translates the linear address supplied by the IU into a physical address to be used by the unified caches and the bus interface. Memory management procedures are x86 compatible, adhering to standard paging mechanisms.

Within the Cyrix III CPU there are two TLBs, the main L1 TLB and the larger L2 TLB. The 16-entry L1 TLB is direct mapped and holds 42 lines. The 384-entry L2 TLB is 6-way associative and hold 384 lines. The DTE is located in memory.

Cache locking is controlled through use of the RDMSR and WRMSR instructions.



1.8 Floating Point Unit

The Floating Point Unit (FPU) processes floating point, MMX, and 3DNOW! instructions. The FPU interfaces to the Integer Unit and the Cache Unit through a 64-bit bus. The FPU is x87 instruction set compatible and adheres to the IEEE-754 standard. Since most applications contain FPU instructions mixed with integer instructions, the FPU achieves high performance by completing integer and FPU operations in parallel.

FPU Parallel Execution

The Cyrix III processor executes integer instructions in parallel with FPU instructions. Integer instructions may complete out of order with respect to the FPU instructions. The Cyrix III processor maintains x86 compatibility by signaling exceptions and issuing write cycles in program order.

FPU instructions can be dispatched from the Integer Unit's X or Y pipeline. The address calculation stage of the pipeline checks for memory management exceptions and accesses memory operands used by the FPU. If no exceptions are detected, the Cyrix III processor checkpoints the state of the CPU and, during AC2, dispatches the floating point instruction to the FPU instruction queue. The Cyrix III processor can then complete any subsequent integer instructions speculatively and out of order relative to the FPU instruction and relative to any potential FPU exceptions which may occur.

As additional FPU instructions enter the pipeline, the Cyrix III processor dispatches up to eight FPU instructions to the FPU instruction queue. The Cyrix III processor continues executing speculatively and out of order, relative to the FPU queue, until the Cyrix III processor encounters one of the conditions that causes speculative execution to halt. As the FPU completes instructions, the speculation level decreases and the checkpointed resources are available for reuse in subsequent operations. The FPU also uses a set of six write buffers to prevent stalls due to speculative writes.

1.9 Bus Interface Unit

The Bus Interface Unit (BIU) provides the signals and timing required by external circuitry. The signal descriptions and bus interface timing information is provided in Chapters 3 and 4 of this manual.

2 PROGRAMMING INTERFACE

In this chapter, the internal operations of the Cyrix III CPU are described mainly from an application programmer's point of view. Included in this chapter are descriptions of processor initialization, the register set, memory addressing, various types of interrupts and the shutdown and halt process. An overview of real, virtual 8086, and protected operating modes is also included in this chapter. The FPU operations are described separately at the end of the chapter.

2.1 Processor Initialization

The Cyrix III CPU is initialized when the RESET# signal is asserted. The processor is placed in real mode and the registers listed in Table 2-1 (Page 2-16) are set to their initialized values. RESET# invalidates and disables the cache and turns off paging. When RESET# is asserted, the Cyrix III CPU terminates all local bus activity and all internal execution. During the entire time that RESET# is asserted, the internal pipelines are flushed and no instruction execution or bus activity occurs.

Approximately 150 to 250 external clock cycles after RESET# is negated, the processor begins executing instructions at the top of physical memory (address location FFFF FFF0h). Typically, an intersegment JUMP is placed at FFFF FFF0h. This instruction will force the processor to begin execution in the lowest 1 MB of address space.

Note: The actual time depends on the clock scaling in use. Also an additional 2^{20} clock cycles are needed if self-test is requested.

Table 2-1. Initialized Core Registers Contents

Register	Register Name	Initialized Contents	Comments
EAX	Accumulator	xxxx xxxxh	0000 0000h indicates self-test passed.
EBX	Base	xxxx xxxxh	
ECX	Count	xxxx xxxxh	
EDX	Data	xxxx 04 [DIR0]	DIR0 = Device ID
EBP	Base Pointer	xxxx xxxxh	
ESI	Source Index	xxxx xxxxh	
EDI	Destination Index	xxxx xxxxh	
ESP	Stack Pointer	xxxx xxxxh	
EFLAGS	Flags	0000 0002h	See Table 2-6 on page 2-26 for bit definitions.
EIP	Instruction Pointer	0000 FFF0h	
ES	Extra Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
CS	Code Segment	F000h	Base address set to FFFF 0000h. Limit set to FFFFh.
SS	Stack Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
DS	Data Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
FS	Extra Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
GS	Extra Segment	0000h	Base address set to 0000 0000h. Limit set to FFFFh.
IDTR	Interrupt Descriptor Table Register	Base = 0, Limit = 3FFh	
GDTR	Global Descriptor Table Register	xxxx xxxxh xxxxh	
LDTR	Local Descriptor Table Register	xxxx xxxxh, xxxxh	
TR	Task Register	xxxxh	
CR0	Machine Status Word	6000 0010h	See Table 2-12 on page29 for bit definitions.
CR2	Control Register 2	xxxx xxxxh	See page30.
CR3	Control Register 3	xxxx xxxxh	See page30.
CR4	Control Register 4	0000 0000h	See Table 2-9 on page 2-30 for bit definitions.
CCR1	Configuration Control 1	00h	See paragraph 2.5.4.1 on page52 for bit definitions.
CCR2	Configuration Control 2	00h	See paragraph 2.5.4.3 on page53 for bit definitions.
CCR3	Configuration Control 3	00h	See paragraph 2.5.4.4 on page54 for bit definitions.
CCR7	Configuration Control 7	00h	See paragraph 2.5.4.8, page58 for bit definitions.
DIR0	Device Identification 0	4xh	Device ID and reads back initial CPU clock-speed setting.
DIR1	Device Identification 1	xxh	Stepping and Revision ID (RO).
DR7	Debug Register 7	0000 0400h	See (XREF) for bit definitions.

Table 2-1. Initialized Core Registers Contents (Continued)

Register	Register Name	Initialized Contents	Comments

x = Undefined value

2.2 Instruction Set Overview

The Cyrix III Processor instruction set can be divided into ten types of operations:

- Arithmetic
- Shift/Rotate
- Control Transfer
- Data Transfer
- Floating Point
- High-Level Language Support
- Operating System Support
- Bit Manipulation
- String Manipulation
- MMX and 3DNow! Instructions

Cyrix III Processor instructions operate on as few as zero operands and as many as three operands. An NOP instruction (no operation) is an example of a zero-operand instruction. Two-operand instructions allow the specification of an explicit source and destination pair as part of the instruction. These two operand instructions can be divided into eight groups according to operand types:

- Register to Register
- Register to Memory
- Memory to Register
- Memory to Memory
- Register to I/O
- I/O to Register
- Immediate Data to Register
- Immediate Data to Memory

An operand can be held in the instruction itself (as in the case of an immediate operand), in one of the processor's registers or I/O ports, or in memory. An immediate operand is fetched as part of the opcode for the instruction.

Operand lengths of 8, 16, 32 or 48 bits are supported as well as 64 or 80 bits associated with floating-point instructions. Operand lengths of 8 or 32 bits are generally used when executing code written for 386- or 486-class (32-bit code) processors. Operand lengths of 8 or 16 bits are generally used when executing existing 8086 or 80286 code (16-bit code). The default length of an operand can be overridden by placing one or more instruction prefixes in front of the opcode. For example, the use of prefixes allows a 32-bit operand to be used with 16-bit code or a 16-bit operand to be used with 32-bit code.

Chapter 6 of this manual lists each instruction in the Cyrix III CPU instruction set along with the associated opcodes, execution clock counts, and effects on the FLAGS register.

2.2.1 Lock Prefix

The LOCK prefix may be placed before certain instructions that read, modify, then write back to memory. The LOCK prefix can be used with the following instructions only when the result is a write operation to memory:

Bit Test Instructions (BTS, BTR, BTC)
Exchange Instructions (XADD, XCHG, CMPXCHG)
One-operand Arithmetic and Logical Instructions (DEC, INC, NEG, NOT)
Two-operand Arithmetic and Logical Instructions (ADC, ADD, AND, OR, SBB, SUB, XOR).

An invalid opcode exception is generated if the LOCK prefix is used with any other instruction or with one of the instructions above when no write operation to memory occurs (for example, when the destination is a register).

2.3 Register Sets

From the programmer's point of view the accessible registers in the Cyrix III CPU are grouped into two sets of registers, the application and system registers set. The application register set contains the registers frequently used by application programmers, and the system register set contains the registers typically reserved for use by operating system programmers.

The application register set is made up of general purpose registers, segment registers, a flag register, and an instruction pointer register.

The system register set is made up of the remaining registers which include control registers, system address registers, debug registers, configuration registers, and test registers.

Each of the registers is discussed in detail in the following sections.

2.3.1 Application Register Set

The Application Register Set, as shown in Table 2-2, consists of the registers most often used by the applications programmer.

These registers are generally accessible, although some bits in the Flags register are protected.

The **General Purpose Register** contents are frequently modified by instructions and typically contain arithmetic and logical instruction operands.

In real mode, **Segment Registers** contain the base address for each segment. In protected mode, the segment registers contain segment selectors. The segment selectors provide indexing for tables (located in memory) that contain the base address for each segment, as well as other memory addressing information.

The **Instruction Pointer Register** points to the next instruction that the processor will execute. This register is automatically incremented by the processor as execution progresses.

The **Flags Register** contains control bits used to reflect the status of previously executed instructions. This register also contains control bits that affect the operation of some instructions.

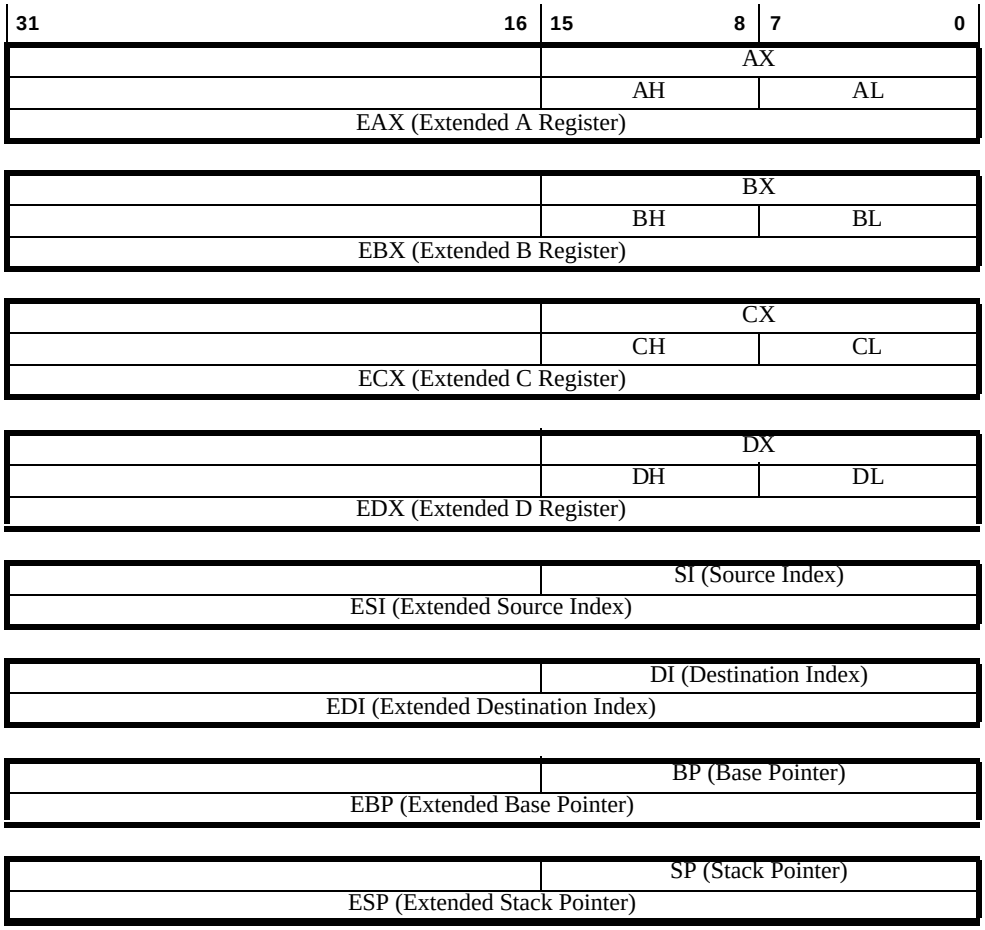
2.3.2 General Purpose Registers

The General Purpose Registers are divided into four data registers, two pointer registers, and two index registers as shown in Table 2-2 on page 2-20.

The **Data Registers** are used by the applications programmer to manipulate data structures and to hold the results of logical and arithmetic operations. Different portions of a general data registers can be addressed by using different names. An “E” prefix identifies a complete 32-bit register. An “X” suffix without the “E” prefix identifies the lower 16 bits of the register.

The lower two bytes of a data register are addressed with an “H” suffix (identifies the upper byte) or an “L” suffix (identifies the lower byte). These _L and _H portions of the data registers act as independent registers. For example, if the AH register is written to by an instruction, the AL register bits remain unchanged.

Table 2-2. Application Register Set
General Purpose Registers



The **Pointer and Index Registers** are listed below.

SI or ESI	Source Index
DI or EDI	Destination Index
SP or ESP	Stack Pointer
BP or EBP	Base Pointer

These registers can be addressed as 16- or 32-bit registers, with the "E" prefix indicating 32 bits.

The pointer and index registers can be used as general purpose registers; however, some instructions use a fixed assignment of these registers. For example, repeated string operations always use ESI as the source pointer, EDI as the destination pointer, and ECX as a counter. The instructions that use fixed registers include multiply and divide, I/O access, string operations, stack operations, loop, variable shift and rotate, and translate instructions.

The Cyrix III Processor implements a stack using the ESP register. This stack is accessed during the PUSH and POP instructions, procedure calls, procedure returns, interrupts, exceptions, and interrupt/exception returns. The Cyrix III Processor automatically adjusts the value of the ESP during operations that result from these instructions.

The EBP register may be used to refer to data passed on the stack during procedure calls. Local data may also be placed on the stack and accessed with BP. This register provides a mechanism to access tack data in high-level languages.

2.3.3 Segment Registers and Selectors

Segmentation provides a means of defining data structures inside the memory space of the microprocessor. There are three basic types of segments: code, data, and stack. Segments are used automatically by the processor to determine the location in memory of code, data, and stack references.

There are six 16-bit segment registers as shown in Table 2-3.

Table 2-3. Application Register Set Segment Selector Registers

15	0
CS (Code Segment)	
SS (Stack Segment)	
DS (D Data Segment)	
ES (E Data Segment)	
FS (F Data Segment)	
GS (G Data Segment)	

In real and virtual 8086 operating modes, a segment register holds a 16-bit segment base. The 16-bit segment is multiplied by 16 and a 16-bit or 32-bit offset is then added to it to create a linear address. The offset size is dependent on the current address size. In real mode and in virtual 8086 mode with paging disabled, the linear address is also the physical address. In virtual 8086 mode with paging enabled, the linear address is translated to the physical address using the current page tables.

Table 2-4. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment	Segment-Override Prefix
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA instructions	SS	None
Source of POP, POPA, POPF, IRET, RET instructions	SS	None
Destination of STOS, MOVS, REP STOS, REP MOVS instructions	ES	None
Other data references with effective address using base registers of:	DS	CS, ES, FS, GS, SS
EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP	SS	CS, DS, ES, FS, GS

In protected mode a segment register holds a Segment Selector containing a 13-bit index, a Table Indicator (TI) bit, and a two-bit Requested Privilege Level (RPL) field. The Index points into a descriptor table in memory and selects one of 8192 (2^{13}) segment descriptors contained in the descriptor table.

A segment descriptor is an eight-byte value used to describe a memory segment by defining the segment base, the segment limit, and access control information. To address data within a segment, a 16-bit or 32-bit offset is added to the segment's base address. Once a segment selector has been loaded into a segment register, an instruction needs only to specify the segment register and the offset.
The Table Indicator (TI) bit of the selector defines which descriptor table the index points into. If TI=0, the index references the Global Descriptor Table (GDT). If TI=1, the index references the Local Descriptor Table (LDT). The GDT and LDT are described in more detail in Section 2.4.2 (Page 2-33). Protected mode addressing is discussed further in Sections 2.8.2 (Page 2-78).

The Requested Privilege Level (RPL) field in a segment selector is used to determine the Effective Privilege Level of an instruction (where RPL=0 indicatesthe most privileged level, and RPL=3 indicatesthe least privileged level).

If the level requested by RPL is less than the Current Program Level (CPL), the RPL level is accepted and the Effective Privilege Level is changed to the RPL value. If the level requested by RPL is greater than CPL, the CPL overrides the requested RPL and Effective Privilege Level remains unchanged.

When a segment register is loaded with a segment selector, the segment base, segment limit and access rights are loaded from the descriptor table entry into a user-invisible or hidden portion of the segment register (i.e., cached on-chip). The CPU does not access the descriptor table entry again until another segment register load occurs. If the descriptor tables are modified in memory, the segment registers must be reloaded with the new selector values by the software.

The active segment register is selected according to the rules listed in Table 2-4 and the type of instruction being currently processed. In general, the DS register selector is used for data references. Stack references use the SS register, and instruction fetches use the CS register. While some of these selections may be overridden, instruction fetches, stack operations, and the destination write operation of string operations cannot be overridden. Special segment-override instruction prefixes allow the use of alternate segment registers. These segment registers include the ES, FS, and GS registers.

2.3.4 Instruction Pointer Register

The Instruction Pointer (EIP) Register contains the offset into the current code segment of the next instruction to be executed. The register is normally incremented by the length of the current instruction with each instruction execution unless it is implicitly modified through an interrupt, exception, or an instruction that changes the sequential execution flow (for example JMP and CALL.

Table 2-5. Application Register Set
Instruction Pointer

31	0
EIP (Extended Instruction Pointer Register)	

2.3.5 Extended Flags Register

The Extended Flags Register, EFLAGS, contains status information and controls certain operations on the Cyrix III CPU microprocessor. The lower 16 bits of this register are referred to as the FLAGS register that is used when executing 8086 or 80286 code. The flag bits listed in Table 2-6.

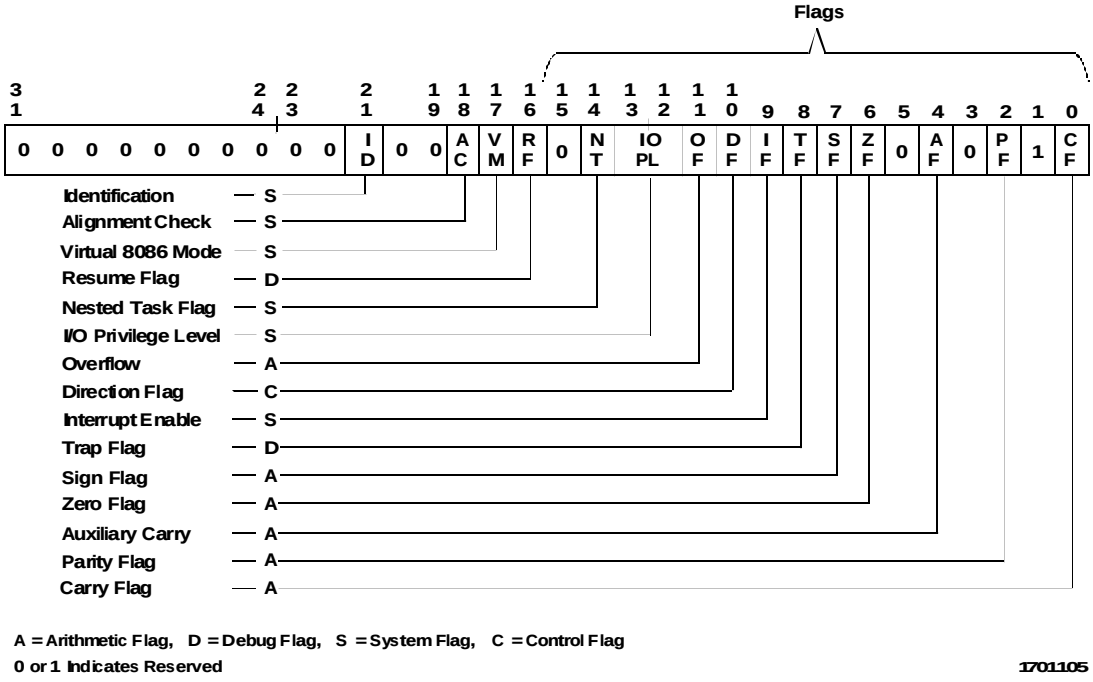


Figure 2-3. EFLAGS Register

Table 2-6. Register Bits EFLAGS Register

Bit	Name	Flag Type	Description
31:22	RSVD	--	Reserved — Set to 0.
21	ID	System	Identification Bit — The ability to set and clear this bit indicates that the CPUID instruction is supported. The ID can be modified only if the CPUID bit in CCR4 (Index E8h[7]) is set.
20:19	RSVD	--	Reserved — Set to 0.
18	AC	System	Alignment Check Enable — In conjunction with the AM flag in CR0, the AC flag determines whether or not misaligned accesses to memory cause a fault. If AC is set, alignment faults are enabled.
17	VM	System	Virtual 8086 Mode — If set while in protected mode, the processor switches to virtual 8086 operation handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes. The VM bit can be set by the IRET instruction (if current privilege level is 0) or by task switches at any privilege level.
16	RF	Debug	Resume Flag — Used in conjunction with debug register breakpoints. RF is checked at instruction boundaries before breakpoint exception processing. If set, any debug fault is ignored on the next instruction.
15	RSVD	--	Reserved — Set to 0.
14	NT	System	Nested Task — While executing in protected mode, NT indicates that the execution of the current task is nested within another task.
13:12	IOPL	System	I/O Privilege Level — While executing in protected mode, IOPL indicates the maximum current privilege level (CPL) permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O permission bit map. IOPL also indicates the maximum CPL allowing alteration of the IF bit when new values are popped into the EFLAGS register.
11	OF	Arithmetic	Overflow Flag — Set if the operation resulted in a carry or borrow into the sign bit of the result but did not result in a carry or borrow out of the high-order bit. Also set if the operation resulted in a carry or borrow out of the high-order bit but did not result in a carry or borrow into the sign bit of the result.
10	DF	Control	Direction Flag — When cleared, DF causes string instructions to auto-increment (default) the appropriate index registers (ESI and/or EDI). Setting DF causes auto-decrement of the index registers to occur.
9	IF	System	Interrupt Enable Flag — When set, maskable interrupts (INTR input pin) are acknowledged and serviced by the CPU.
8	TF	Debug	Trap Enable Flag — Once set, a single-step interrupt occurs after the next instruction completes execution. TF is cleared by the single-step interrupt.
7	SF	Arithmetic	Sign Flag — Set equal to high-order bit of result (0 indicates positive, 1 indicates negative).
6	ZF	Arithmetic	Zero Flag — Set if result is zero; cleared otherwise.
5	RSVD	--	Reserved — Set to 0.
4	AF	Arithmetic	Auxiliary Carry Flag — Set when a carry out of (addition) or borrow into (subtraction) bit position 3 of the result occurs; cleared otherwise.
3	RSVD	--	Reserved — Set to 0.

Table 2-6. Register Bits EFLAGS Register (Continued)

Bit	Name	Flag Type	Description
2	PF	Arithmetic	Parity Flag — Set when the low-order 8 bits of the result contain an even number of ones; otherwise PF is cleared.
1	RSVD		Reserved — Set to 1.
0	CF	Arithmetic	Carry Flag — Set when a carry out of (addition) or borrow into (subtraction) the most significant bit of the result occurs; cleared otherwise.

2.4 System Register Set

The system register set is used for system level programming. The system register set consists of registers not generally used by application programmers. These registers are typically employed by system level programmers who generate operating systems and memory management programs. Associated with the system register set are tables and segments which are defined in memory.

The Control Registers control certain aspects of the Cyrix III Processor such as paging, coprocessor functions, and segment protection.

The Descriptor Tables hold descriptors that manage memory segments and tables, interrupts and task switching. The tables are defined by corresponding registers.

The two **Task State Segments Tables** defined by TSS register, are used to save and load the computer state when switching tasks.

The Configuration Registers are used to define Cyrix III Processor CPU setup including cache management.

The **ID Registers** allow BIOS and other software to identify the specific CPU and stepping. System Management Mode (SMM) control information is stored in the SMM registers.

The Debug Registers provide debugging facilities for the Cyrix III Processor and enable the use of data access breakpoints and code execution breakpoints.

The Test Registers provide a mechanism to test the contents of both the on-chip 16KB cache and the Translation Lookaside Buffer (TLB). The TLB is used as a cache for the tables that are used in to translate linear addresses to physical addresses while paging is enabled

2.4.1 Control Registers

The standard x86 Control Registers (CR0, CR2, CR3 and CR4), are shown in Table 2-7.[1] The CR0 register contains system control bits which configure operating modes and indicate the general state of the CPU. The lower 16 bits of CR0 are referred to as the Machine Status Word (MSW). The CR0 bit definitions are described in Table 2-13. The reserved bits in CR0 should not be modified. A CR1 register is not defined.

When paging is enabled and a page fault is generated, the CR2 register retains the 32-bit linear address of the address that caused the fault. When a double page fault occurs, CR2 contains the address for the second fault. Register CR3 contains the 20 most significant bits of the physical base address of the page directory. The page directory must always be aligned to a 4-KB page boundary, therefore, the lower 12 bits of CR3 are not required to specify the base address.

Register CR3 contains the 20 most significant bits of the physical base address of the page directory. The page directory must always be aligned to a 4KB page boundary, therefore, the lower 12 bits of CR3 are not required to specify the base address.

CR3 also contains the Page Cache Disable (PCD) and Page Write Through (PWT) bits. Control Register CR4 Table 2-9 on page 30 controls usage of the Time Stamp Counter Instruction, Debugging Extensions, Page Global Enable and the RDPMC instruction.

1. The CRn are standard x86 registers, and are distinct from the CCRn registers unique to Cyrix.)

Table 2-7. Control Registers

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

CR4 Register

RSVD	PCE	PGE	RSVD	DE	TSC	RSVD

CR3 Register

PDBR (Page Directory Base Register)	RSVD	PCD	PWT	RSVD

CR2 Register

PFLA (Page Fault Linear Address)

CR1 Register

RSVD

CR0 Register

PG	CD	NW	RSVD	AM	RSVD	WP	RSVD	NE	1	TS	EM	MP	PE
							Machine Status Word (MSW)						

Table 2-8. CR4 Bit Definitions

BIT POSITION	NAME	FUNCTION
2	TSD	Time Stamp Counter Instruction If = 1 RDTSC instruction enabled for CPL=0 only; Reset State If = 0 RDTSC instruction enabled for all CPL states
3	DE	Debugging Extensions If = 1 enables I/O breakpoints and R/W bits for each debug register are defined as: 00 -Break on instruction execution only. 01 -Break on data writes only. 10 -Break on I/O reads or writes. 11 -Break on data reads or writes but not instruction fetches. If = 0 I/O breakpoints and R/W bits for each debug register are not enabled.
7	PGE	Page Global Enable If = 1 global page feature is enabled. If = 0 global page feature is disabled. Global pages are not flushed from TLB on a task switch or write to CR3
8	PCE	Performance Monitoring Counter Enable If = 1 enables execution of RDPMC instruction at any protection level. If = 0 RDPMC instruction can only be executed at protection level 0.

Table 2-9. CR3 Bit Definitions

Bits	Name	Description
31 - 12	PDBR	Page Directory Base Register: Identifies page directory base address on a 4KB page boundary.
11 - 5	RSVD	Reserved**:** Set to 0.
4	PCD	**Page Cache Disable:** During bus cycles that are not paged, the state of the PCD bit is reflected on the PCD pin. These bus cycles include interrupt acknowledge cycles and all bus cycles, when paging is not enabled. The PCD pin should be used to control caching in an external cache.
3	PWT	**Page Write-Through:** During bus cycles that are not paged, the state of the PWT bit is driven on the PWT pin. These bus cycles include interrupt acknowledge cycles and all bus cycles, when paging is not enabled. The PWT pin should be used to control write policy in an external cache.
2-1	RSVD	Reserved**:** Set to 0.

Table 2-10. CR2 Bit Definitions

Bits	Name	Description
31 - 0	PFLA	Page Fault Linear Address: With paging enabled and after a page fault, PFLA contains the linear address of the address that caused the page fault.

Table 2-11. CR1 Bit Definitions

Bit	Name	Description
31:0	RSVD	**Reserved:** Set to 0 (always returns 0 when read).

Table 2-12. CR0 Bit Definitions

Bit	Name	Description
31	PG	**Paging Enable Bit:** If PG=1 and protected mode is enabled (PE=1), paging is enabled. After changing the state of PG, software must execute an unconditional branch instruction (e.g., JMP, CALL) to have the change take effect.
30	CD	**Cache Disable:** If CD=1, no further cache line fills occur. However, data already present in the cache continues to be used if the requested address hits in the cache. Writes continue to update the cache and cache invalidations due to inquiry cycles occur normally. The cache must also be invalidated to completely disable any cache activity.
29	NW	**Not Write-Back:** If NW=1, the on-chip cache operates in write-through mode. In write-through mode, all writes (including cache hits) are issued to the external bus. If NW=0, the on-chip cache operates in write-back mode. In write-back mode, writes are issued to the external bus only for a cache miss, a line replacement of a modified line, or as the result of a cache inquiry cycle.
28:19	RSVD	**Reserved:** Do not modify.
18	AM	Alignment Check Mask: If AM=1, the AC bit in the EFLAGS register is unmasked and allowed to enable alignment check faults. Setting AM=0 prevents AC faults from occurring.
17	RSVD	**Reserved:** Do not modify.
16	WP	Write Protect: Protects read-only pages from supervisor write access. WP=0 allows a read-only page to be written from privilege level 0-2. WP=1 forces a fault on a write to a read-only page from any privilege level.
15:6	RSVD	**Reserved:** Do not modify.
5	NE	**Numerics Exception:** NE=1 to allow FPU exceptions to be handled by interrupt 16. NE=0 if FPU exceptions are to be handled by external interrupts.
4	1	**Reserved:** Do not attempt to modify.
3	TS	**Task Switched:** Set whenever a task switch operation is performed. Execution of a floating point instruction with TS=1 causes a DNA fault. If MP=1 and TS=1, a WAIT instruction also causes a DNA fault.
2	EM	**Emulate Processor Extension:** If EM=1, all floating point instructions cause a DNA fault 7.
1	MP	**Monitor Processor Extension:** If MP=1 and TS=1, a WAIT instruction causes Device Not Available (DNA) fault 7. The TS bit is set to 1 on task switches by the CPU. Floating point instructions are not affected by the state of the MP bit. The MP bit should be set to one during normal operations.
0	PE	**Protected Mode Enable:** Enables the segment based protection mechanism. If PE=1, protected mode is enabled. If PE=0, the CPU operates in real mode and addresses are formed as in an 8086-style CPU.

Table 2-13. CR0 Register EM, TS, and MP Bits Combinations

CR0 Register Bits			Instruction Type	
EM Bit 2	TS Bit 3	MP Bit 1	WAIT	ESC
0	0	0	Execute	Execute
0	0	1	Execute	Execute
0	1	0	Execute	Fault 7
0	1	1	Fault 7	Fault 7
1	0	0	Execute	Fault 7
1	0	1	Execute	Fault 7
1	1	0	Execute	Fault 7
1	1	1	Fault 7	Fault 7

2.4.2 Descriptor Table Registers and Descriptors

Descriptor Table Registers

The Global, Interrupt, and Local Descriptor Table Registers (GDTR, IDTR and LDTR), shown in Figure 2-4, are used to specify the location of the data structures that control segmented memory management. The GDTR, IDTR and LDTR are loaded using the LGDT, LIDT and LLDT instructions, respectively. The values of these registers are stored using the corresponding store instructions. The GDTR and IDTR load instructions are privileged instructions when operating in protected mode. The LDTR can only be accessed in protected mode.

The Global Descriptor Table Register (GDTR) holds a 32-bit linear base address and 16-bit limit for the Global Descriptor Table (GDT). The GDT is an array of up to 8192 8-byte descriptors. When a segment register is loaded from memory, the TI bit in the segment selector chooses either the GDT or the Local Descriptor Table (LDT) to locate a descriptor. If TI = 0, the index portion of the selector is used to locate the descriptor within the GDT table. The contents of the GDTR are completely visible to the programmer by using a SGDT instruction. The first descriptor in the GDT (location 0) is not used by the CPU and is referred to as the “null descriptor”. The GDTR is initialized using a LGDT instruction.

The Interrupt Descriptor Table Register (IDTR) holds a 32-bit linear base address and 16-bit limit for the Interrupt Descriptor Table (IDT). The IDT is an array of 256 interrupt descriptors, each of which is used to point to an interrupt service routine. Every interrupt that may occur in the system must have an associated entry in the IDT. The contents of the IDTR are completely visible to the programmer by using a SIDT instruction. The IDTR is initialized using the LIDT instruction.

The Local Descriptor Table Register (LDTR) holds a 16-bit selector for the Local Descriptor Table (LDT). The LDT is an array of up to 8192 8-byte descriptors. When the LDTR is loaded, the LDTR selector indexes an LDT descriptor that resides in the Global Descriptor Table (GDT). The base address and limit are loaded automatically and cached from the LDT descriptor within the GDT.

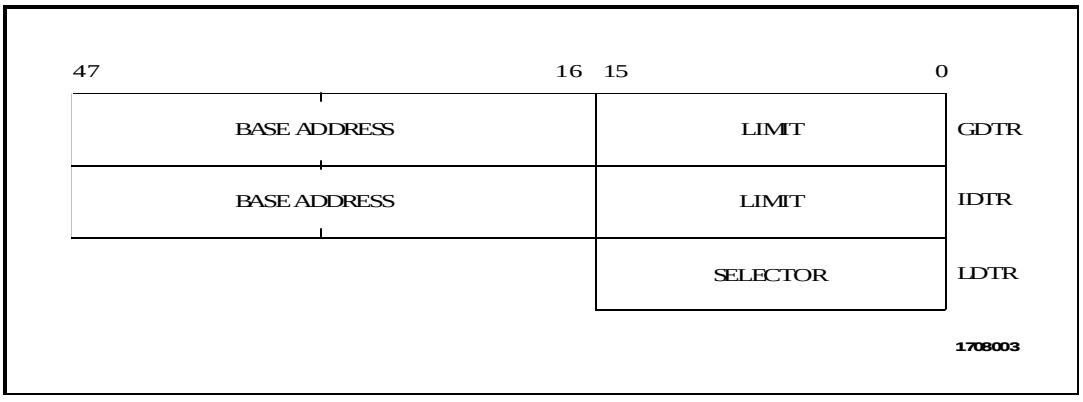


Figure 2-4. Descriptor Table Registers

Subsequent access to entries in the LDT use the hidden LDTR cache to obtain linear addresses. If the LDT descriptor is modified in the GDT, the LDTR must be reloaded to update the hidden portion of the LDTR.

When a segment register is loaded from memory, the TI bit in the segment selector chooses either the GDT or the LDT to locate a segment descriptor. If TI = 1, the index portion of the selector is used to locate a given descriptor within the LDT. Each task in the system may be given its own LDT, managed by the operating system. The LDTs provide a method of isolating a given task's segments from other tasks in the system.

The LDTR can be read or written by the LLDT and SLDT instructions.

Descriptors

There are three types of descriptors:

- Application Segment Descriptors that define code, data and stack segments.
- System Segment Descriptors that define an LDT segment or a Task State Segment (TSS) table described later in this text.
- Gate Descriptors that define task gates, interrupt gates, trap gates and call gates.

Application Segment Descriptors can be located in either the LDT or GDT. System Segment Descriptors can only be located in the GDT. Dependent on the gate type, gate descriptors may be located in either the GDT, LDT or IDT. Figure 2-5 illustrates the descriptor format for both Application Segment Descriptors and System Segment Descriptors. Table 2-14 (Page 2-35) lists the corresponding bit definitions.

Table 2-15. (Page 2-35) and Table 2-16. (Page 2-36) defines the DT field within the segment descriptor.

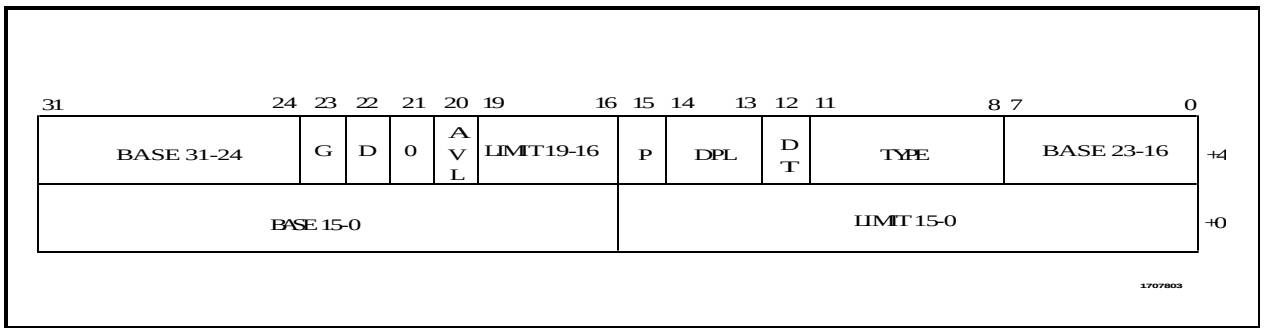


Figure 2-5. Application and System Segment Descriptors

Table 2-14. Segment Descriptor Bit Definitions

BIT POSITION	MEMORY OFFSET	NAME	DESCRIPTION
31-24 7-0 31-16	+4 +4 +0	BASE	Segment base address. 32-bit linear address that points to the beginning of the segment.
19-16 15-0	+4 +0	LIMIT	Segment limit.
23	+4	G	Limit granularity bit: 0 = byte granularity, 1 = 4 KB (page) granularity.
22	+4	D	Default length for operands and effective addresses. Valid for code and stack segments only: 0 = 16 bit, 1 = 32-bit.
20	+4	AVL	Segment available.
15	+4	P	Segment present.
14-13	+4	DPL	Descriptor privilege level.
12	+4	DT	Descriptor type: 0 = system, 1 = application.
11-8	+4	TYPE	Segment type. See Tables 2-7 and 2-8.

Table 2-15. TYPE Field Definitions with DT = 0

TYPE (BITS 11-8)	DESCRIPTION
0001	TSS-16 descriptor, task not busy.
0010	LDT descriptor.
0011	TSS-16 descriptor, task busy.
1001	TSS-32 descriptor, task not busy
1011	TSS-32 descriptor, task busy.

Table 2-16. TYPE Field Definitions with DT = 1

TYPE				APPLICATION DECRIPTOR INFORMATION
E	C/D	R/W	A	
0	0	x	x	data, expand up, limit is upper bound of segment
0	1	x	x	data, expand down, limit is lower bound of segment
1	0	x	x	executable, non-conforming
1	1	x	x	executable, conforming (runs at privilege level of calling procedure)
0	x	0	x	data, non-writable
0	x	1	x	data, writable
1	x	0	x	executable, non-readable
1	x	1	x	executable, readable
x	x	x	0	not-accessed
x	x	x	1	accessed

Gate Descriptors provide protection for executable segments operating at different privilege levels. Figure 2-7 illustrates the format for Gate Descriptors and the table lists the corresponding bit definitions.

Task Gate Descriptors are used to switch the CPU's context during a task switch. The selector portion of the task gate descriptor locates a Task State Segment. These descriptors can be located in the GDT, LDT or IDT tables.

Interrupt Gate Descriptors are used to enter a hardware interrupt service routine. Trap Gate Descriptors are used to enter exceptions or software interrupt service routines. Trap Gate and Interrupt Gate Descriptors can only be located in the IDT.

Call Gate Descriptors are used to enter a procedure (subroutine) that executes at the same or a more privileged level. A Call Gate Descriptor primarily defines the procedure entry point and the procedure's privilege level.

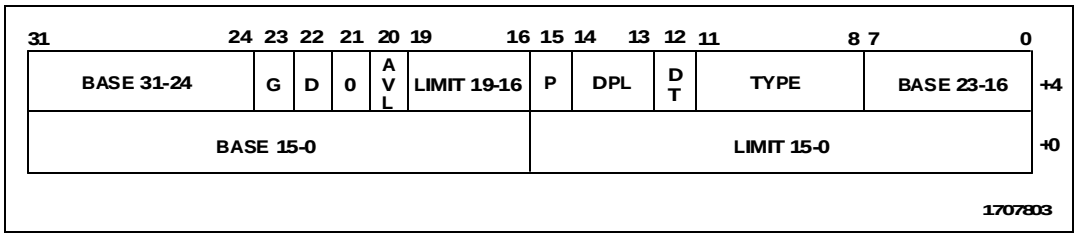


Figure 2-6 Gate Descriptor

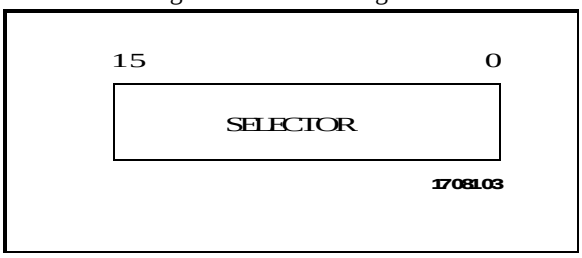
BIT POSITION	MEMORY OFFSET	NAME	DESCRIPTION
31-16 15-0	+4 +0	OFFSET	Offset used during a call gate to calculate the branch target.
31-16	+0	SELECTOR	Segment selector used during a call gate to calculate the branch target.
15	+4	P	Segment present.
14-13	+4	DPL	Descriptor privilege level.
11-8	+4	TYPE	Segment type: 0100 = 16-bit call gate 0101 = task gate 0110 = 16-bit interrupt gate 0111 = 16-bit trap gate 1100 = 32-bit call gate 1110 = 32-bit interrupt gate 1111 = 32-bit trap gate.
4-0	+4	PARAME-TERS	Number of 32-bit parameters to copy from the caller's stack to the called procedure's stack (valid for calls).

2.4.3 Task Register

The Task Register (TR) holds a 16-bit selector for the current Task State Segment (TSS) table as shown in Figure 2-7. The TR is loaded and stored via the LTR and STR instructions, respectively. The TR can only be accessed during protected mode and can only be loaded when the privilege level is 0 (most privileged). When the TR is loaded, the TR selector field indexes a TSS descriptor that must reside in the Global Descriptor Table (GDT). The contents of the selected descriptor are cached on-chip in the hidden portion of the TR.

During task switching, the processor saves the current CPU state in the TSS before starting a new task. The TR points to the current TSS. The TSS can be either a 386/486-style 32-bit TSS or a 286-style 16-bit TSS type. An I/O permission bit map is referenced in the 32-bit TSS by the I/O Map Base Address.

Figure 2-7. Task Register



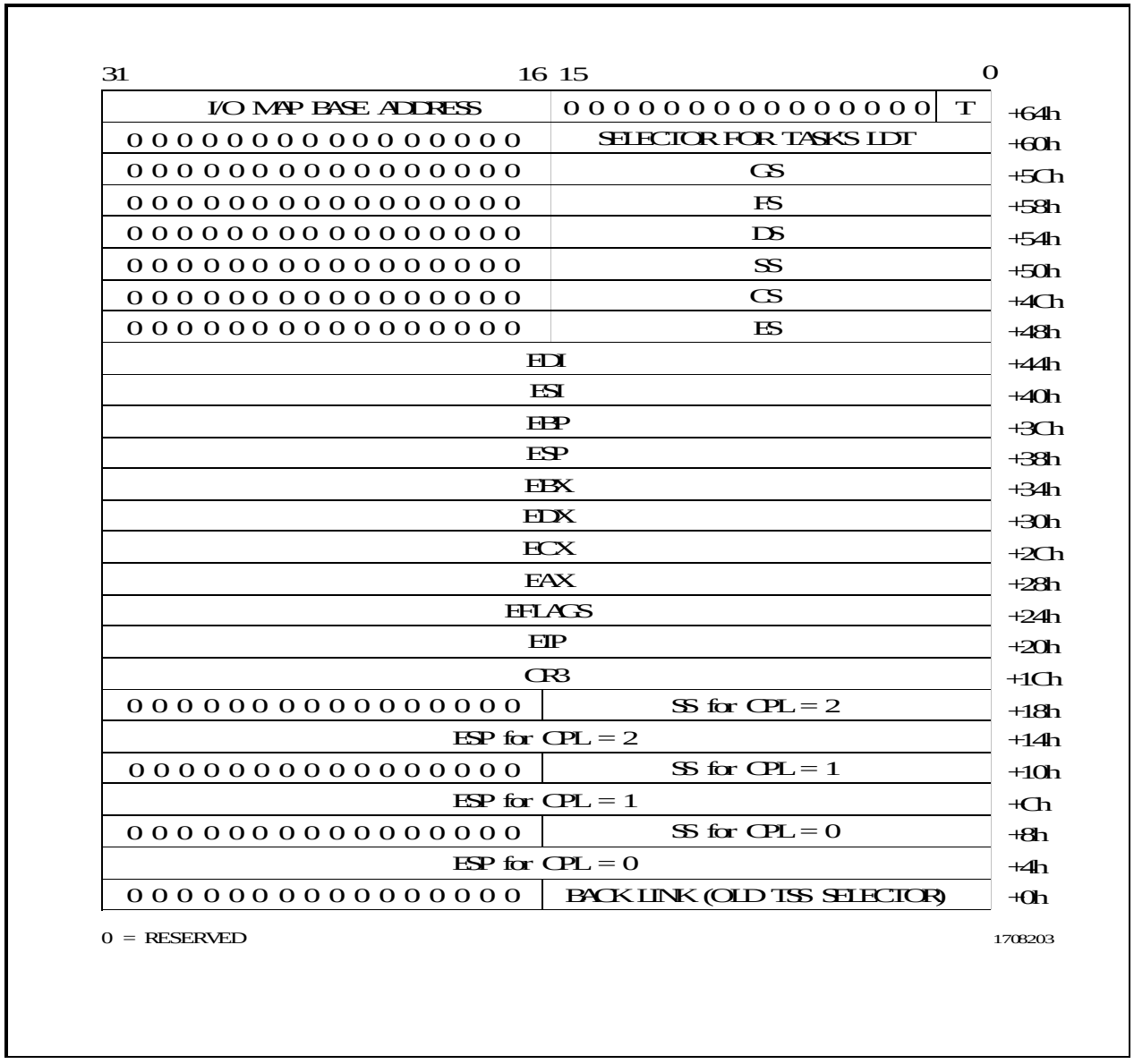


Figure 2-8. 32-Bit Task State Segment (TSS) Table

System Register Set

SELECTOR FOR TASK'S LDT	+2Ah
DS	+28h
SS	+26h
CS	+24h
ES	+22h
DI	+20h
SI	+1Eh
BP	+1Ch
SP	+1Ah
BX	+18h
DX	+16h
CX	+14h
AX	+12h
FLAGS	+10h
IP	+Eh
SS FOR PRIVILEGE LEVEL 2	+Ch
SP FOR PRIVILEGE LEVEL 2	+Ah
SS FOR PRIVILEGE LEVEL 1	+8h
SP FOR PRIVILEGE LEVEL 1	+6h
SS FOR PRIVILEGE LEVEL 0	+4h
SP FOR PRIVILEGE LEVEL 0	+2h
BACK LINK (OLD TSS SELECTOR)	+0h

1708803

Table 2-17. 16-Bit Task State Segment (TSS) Table

2.4.4 Model Specific Registers

The CPU contains several Model Specific Registers (MSR's) that provide time stamp, performance monitoring and counter event functions. Access to a specific MSR through an index value in the ECX register as shown in Table 18 below.

Table 2-18. Model Specific Register

Register Description	ECX Value
Test Data	3h
Test Address	4h
Command/Status	5h
Time Stamp Counter (TSC)	10h
Counter Event Control Register	11h
Performance Counter #0	12h
Performance Counter #1	13h

The MSR registers can be read using the RDMSR instruction, opcode 0F32h. During an MSR register read, the contents of the particular MSR register, specified by the ECX register, is loaded into the EDX:EAX registers.

The MSR registers can be written using the WRMSR instruction, opcode 0F30h. During a MSR register write the contents of EDX:EAX are loaded into the MSR register specified in the ECX register.

The RDMSR and WRMSR instructions are privileged instructions and are also used to setup scratchpad lock.

2.4.4.1 Time Stamp Counter

The Time Stamp Counter (TSC) Register MSR[10] is a 64-bit counter that counts the internal CPU clock cycles since the last reset. The TSC uses a continuous CPU core clock and will continue to count clock cycles even when the processor is in Suspend mode.

The TSC can be accessed using the RDMSR and WRMSR instructions. In addition, the TSC can be read using the RDTSC instruction, opcode 0F31h. The RDTSC instruction loads the contents of the TSC into EDX:EAX. The use of the RDTSC instruction is restricted by the Time Stamp Counter, (TSC) flag in CR4. When the TSC flag is 0, the RDTSC instruction can be executed at any privilege level. When the TSC flag is 1, the RDTSC instruction can only be executed at privilege level0.

2.4.4.2 Performance Monitoring

Performance monitoring allows counting of over a hundred different event occurrences and durations. Two 48-bit counters are used: Performance Monitor Counter 0 and Performance Monitor Counter 1. These two performance monitor counters are controlled by the Counter Event Selection and Control Register MSR[11]. The performance monitor counters use a continuous CPU core clock and will continue to count clock cycles even when the processor is in Suspend or Shutdown mode.

2.4.4.2.1 Performance Monitoring Counters1and2

The 48-bit Performance Monitoring Counter (PMC) Registers MSR[12] and MSR[13] count events as specified by the Counter Event Selection and Control Register, MSR[11].

The PMCs can be accessed by the RDMSR and WRMSR instructions. In addition, the PMCs can be read by the RDPMC instruction, opcode 0F33h. The RDPMC instruction loads the contents of the PMC register specified in the ECX register into EDX:EAX. The use of RDPMC instructions is restricted by the Performance Monitoring Counter Enable, (PCE) flag in CR4.

When the PCE flag is set to 1, the RDPMC instruction can be executed at any privilege level. When the PCE flag is 0, the RDPMC instruction can only be executed at privilege level 0.

2.4.4.2.2 Counter Event Selection and Control Register

Register MSR[11] controls the two internal counters, #0 and #1. The events to be counted have been chosen based on the micro-architecture of the Cyrix III processor. The control register for the two event counters is described in Table 2-19.

2.4.4.2.3 Counter Type Control

The Counter Type bit determines whether the counter will count clocks or events. When counting clocks the counter operates as a timer.

2.4.4.2.4 CPL Control

The Current Privilege Level (CPL) can be used to determine if the counters are enabled. The CP02 bit in the MSR[11] register enables counting when the CPL is less than three, and the CP03 bit enables counting when CPL is equal to three. If both bits are set, counting is not dependent on the CPL level; if neither bit is set, counting is disabled.

Table 2-19. Counter Event Selection and Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RSVD					TC1*	RSVD	CT1	CP13	CP12	TC1*						RSVD					TC0*	RSVD	CT0	CP03	CP02	TC0*					

Note: Split fields.

Table 2-20. Counter Event Selection and Control Register Bit Definitions

Bit	Name	Description
32:27	RSVD	Reserved
25	RSVD	Reserved
24	CT1	**Counter #1 Counter Type:** If = 1: Count clock cycles If = 0: Count events (reset state).
23	CP13	**Counter #1 CPL 3 Enable:** If = 1: Enable counting when CPL=3. If = 0: Disable counting when CPL=3. (Reset state)
22	CP12	**Counter #1 CPL Less Than 3 Enable:** If = 1: Enable counting when CPL < 3. If = 0: Disable counting when CPL < 3. (Reset state)
26, 21:16	TC1[5:0]	**Counter #1 Event Type:** Reset state = 0
15:9	RSVD	Reserved
8	CT0	**Counter #0 Counter Type**: If = 1: Count clock cycles If = 0: Count events (reset state).
7	CP03	**Counter #0 CPL 3 Enable:** If = 1: Enable counting when CPL=3. If = 0: Disable counting when CPL=3. (reset state)
6	CP02	**Counter #0 CPL Less Than 3 Enable:** If = 1: Enable counting when CPL < 3. If = 0: Disable counting when CPL < 3. (reset state)
10, 5:0	TC0[5:0]	**Counter #0 Event Type:** Reset state = 0

2.4.5 Event Type and Description

The events that can be counted by the performance monitoring counters are listed in Table 2-21. Each of the 127 event types is assigned an event number.

A particular event number to be counted is placed in one of the MSR[11] Event Type fields. There is a separate field for counter #0 and #1.

The events are divided into two groups. The occurrence type events and duration type events. The occurrence type events, such as hardware interrupts, are counted as single events. The duration type events such as “clock while bus cycles are in progress” count the number of clock cycles that occur during the event.

Table 2-21. Event Type Register

Number	Counter #0	Counter #1	Description	Type
00h	Yes	Yes	Data reads	Occurrence
01h	Yes	Yes	Data writes	Occurrence
02h	Yes	Yes	Data TLB misses	Occurrence
03h	Yes	Yes	Cache misses: Data reads	Occurrence
04h	Yes	Yes	Cache misses: Data writes	Occurrence
05h	Yes	Yes	Data writes that hit on modified or exclusive lines	Occurrence
06h	Yes	Yes	Data cache lines written back	Occurrence
07h	Yes	Yes	External inquiries	Occurrence
08h	Yes	Yes	External inquires that hit	Occurrence
09h	Yes	Yes	Memory accesses in both pipelines	Occurrence
0Ah	Yes	Yes	Cache bank conflicts	Occurrence
0Bh	Yes	Yes	Misaligned data references	Occurrence
0Ch	Yes	Yes	Instruction fetch requests	Occurrence
0Dh	Yes	Yes	L2 TLB code misses	Occurrence
0Eh	Yes	Yes	Cache misses: Instruction fetch	Occurrence
0Fh	Yes	Yes	Any Segment Register load	Occurrence
10h	Yes	Yes	Reserved	Occurrence
11h	Yes	Yes	Reserved	Occurrence
12h	Yes	Yes	Any branch	Occurrence
13h	Yes	Yes	BTB hits	Occurrence
14h	Yes	Yes	Taken branches or BTB hits	Occurrence
15h	Yes	Yes	Pipeline flushes	Occurrence
16h	Yes	Yes	Instructions executed in both pipelines	Occurrence
17h	Yes	Yes	Instructions executed in Y pipeline	Occurrence
18h	Yes	Yes	Clocks while bus cycles are in progress	Duration
19h	Yes	Yes	Pipeline stalled by full write buffers	Duration
1Ah	Yes	Yes	Pipeline stalled by waiting on data memory reads	Duration
1Bh	Yes	Yes	Pipeline stalled by writes to not-modified or not-exclusive cache lines.	Duration

Table 2-21. Event Type Register (Continued)

Number	Counter #0	Counter #1	Description	Type
1Ch	Yes	Yes	Locked bus cycles	Occurrence
1Dh	Yes	Yes	I/O cycles	Occurrence
1Eh	Yes	Yes	Non-cacheable memory requests	Occurrence
1Fh	Yes	Yes	Pipeline stalled by address generation interlock	Duration
20h	Yes	Yes	Reserved	--
21h	Yes	Yes	Reserved	--
22h	Yes	Yes	Floating point operations	Occurrence
23h	Yes	Yes	Breakpoint matches on DR0 register	Occurrence
24h	Yes	Yes	Breakpoint matches on DR1 register	Occurrence
25h	Yes	Yes	Breakpoint matches on DR2 register	Occurrence
26h	Yes	Yes	Breakpoint matches on DR3 register	Occurrence
27h	Yes	Yes	Hardware interrupts	Occurrence
28h	Yes	Yes	Data reads or data writes	Occurrence
29h	Yes	Yes	Data read misses or data write misses	Occurrence
2Bh	Yes	No	MMX instruction executed in X pipeline	Occurrence
2Bh	No	Yes	MMX instruction executed in Y pipeline	Occurrence
2Dh	Yes	No	EMMS instruction executed	Occurrence
2Dh	No	Yes	Transition between MMX instruction and FP instructions	Occurrence
2Eh	No	Yes	Reserved	--
2Fh	Yes	No	Saturating MMX instructions executed	Occurrence
2Fh	No	Yes	Saturations performed	Occurrence
30h	Yes	No	Reserved	--
31h	Yes	No	MMX instruction data reads	Occurrence
32h	Yes	No	Reserved	--
32h	No	Yes	Taken branches	Occurrence
33h	No	Yes	Reserved	--
34h	Yes	No	Reserved	--
34h	No	Yes	Reserved	--
35h	Yes	No	Reserved	--
35h	No	Yes	Reserved	--
36h	Yes	No	Reserved	--
36h	No	Yes	Reserved	--
37h	Yes	No	Returns predicted incorrectly	Occurrence
37h	No	Yes	Return predicted (correctly and incorrectly)	Occurrence
38h	Yes	No	MMX instruction multiply unit interlock	Duration
38h	No	Yes	MODV/MOVQ store stall due to previous operation	Duration
39h	Yes	No	Returns	Occurrence
39h	No	Yes	RSB overflows	Occurrence
3Ah	Yes	No	BTB false entries	Occurrence
3Ah	No	Yes	BTB miss prediction on a not-taken back	Occurrence
3Bh	Yes	No	Number of clock stalled due to full write buffers while executing	Duration

Table 2-21. Event Type Register (Continued)

Number	Counter #0	Counter #1	Description	Type
3Bh	No	Yes	Stall on MMX instruction write to E or M line	Duration
3Ch-3Fh	Yes	Yes	Reserved	--
40h	Yes	Yes	L2 TLB misses (code or data)	Occurrence
41h	Yes	Yes	L1 TLB data miss	Occurrence
42h	Yes	Yes	L1 TLB code miss	Occurrence
43h	Yes	Yes	L1 TLB miss (code or data)	Occurrence
44h	Yes	Yes	TLB flushes	Occurrence
45h	Yes	Yes	TLB page invalidates	Occurrence
46h	Yes	Yes	TLB page invalidates that hit	Occurrence
47h	Yes	Yes	Reserved	--
48h	Yes	Yes	Instructions decoded	Occurrence
49h	Yes	Yes	Reserved	--

2.5 Cyrix III Register Set

The Cyrix III Register Set includes the configuration registers that are used to enable features in the CPU. These features are specific to the Cyrix III processor architecture and are typically only accessed during the boot/initialization process.

Registers included in this section are:

- Configuration Control Registers
- Address Region Registers
- Region Control Registers

2.5.1 I/O Port 22h and 23h Access

Access to internal registers is accomplished via an 8-bit index/data I/O pair. Each data transfer, I/O Port 23h, must be preceded by a valid index write, I/O Port 22h. All reads from I/O Port 22h produce external I/O cycles; therefore, the index cannot be read. Accesses that hit within the on-chip configuration registers do not generate external I/O cycles.

To access the above registers, the programmer uses the Port 22h (index) and Port 23h (data). The access is atomic when an SMI is involved, but is not atomic when any of the following three conditions are presented: 1) INT, 2) NMI 3) INIT#. Proper steps must be taken to inhibit these three conditions if a pure atomic operation is to be achieved. An example of this to prevent an INT sequence would be to use a PUSHF, CLI instruction pair before doing the Port 22h/23h access with a POPF after the access has been done.

After reset, only configuration registers with indices C0-CFh and FC-FFh are accessible. This prevents potential conflicts with other devices that use Ports 22h and 23h to access their registers.

2.5.2 Map Enable (MAPEN)

The purpose of MAPEN is to increase the number of registers that can be accessed via Ports 22h/23h. The MAPEN fields can be thought of as a paging mechanism to the complete register set allowing multiple registers to share the same index value but providing different functionality. MAPEN must be set correctly to gain access to the register that is to be modified. MAPEN is located in CCR3[7:4]. It is strongly recommended that the programmer use the following sequence:

1) Read CCR3.
2) Save CCR3.
3) Modify MAPEN at CCR3[7-4].
4) Access Register via Port 22h/23h access.
5) Restore CCR3 to control the MAPEN field.

There are 256 possible registers available for each MAPEN setting, for a total of 4096. Most registers are not defined and are therefore reserved.

2.5.3 Cyrix Configuration Control Registers

Table 2-22 summarizes the Core Registers. The registers are described in greater detail beginning on page 52.

Table 2-22. CPU Configuration Register Summary

Acronym	Register	Index	Size (Bits)	MAPEN CCR3[7-4]	Reference
CCR0	Configuration Control 0	C0h	8	xxxxb	
CCR1	Configuration Control 1	C1h			
CCR2	Configuration Control 2	C2h			
CCR3	Configuration Control 3	C3h			
CCR4	Configuration Control 4	E8h		0001b	
CCR5	Configuration Control 5	E9h			
CCR6	Configuration Control 6	EAh			
CCR7	Configuration Control 7	EBh			
ARR0	Address Region 0	C4h-C6h	24	xxxxb	
ARR1	Address Region 1	C7h-C9h			
ARR2	Address Region 2	CAh-CCh			
ARR3	Address Region 3	CDh-CFh			
ARR4	Address Region 4	Dh0-D2h		0001b	
ARR5	Address Region 5	D3h-D5h			
ARR6	Address Region 6	D6h-D8h			
ARR7	Address Region 7	D9h-DBh			
ARR8	Address Region 8	A4h-A6h		read= x010b write=x01xb	
ARR9	Address Region 9	A7h-A9h			
ARRA	Address Region A	AAh-ACh			
ARRB	Address Region B	ADh-AFh			
ARRC	Address Region C	D0h-D2h			
ARRD	Address Region D	D3h-D5h			
RCR0	Region Configuration Register 0	DCh	8	0001b	
RCR1	Region Configuration Register 1	DDh			

Table 2-22. CPU Configuration Register Summary (Continued)

Acronym	Register	Index	Size (Bits)	MAPEN CCR3[7-4]	Reference
RCR2	Region Configuration Register 2	DEh	8	x00xb	
RCR3	Region Configuration Register 3	DFh			
RCR4	Region Configuration Register 4	E0h			
RCR5	Region Configuration Register 5	E1h			
RCR6	Region Configuration Register 6	E2h			
RCR7	Region Configuration Register 7	E3h			
RCR8	Region Configuration Register 8	DCh			
RCR9	Region Configuration Register 9	DDh			
RCRA	Region Configuration Register A	DEh			
RCRB	Region Configuration Register B	DFh			
RCRC	Region Configuration Register C	E0h			
RCRD	Region Configuration Register D	E1h			
DIR0	Directory Register 0	FEh		xxxxb	
DIR1	Directory Register 1	FFh			
DIR2	Directory Register 2	FDh			
DIR3	Directory Register 3	FCh		read= x010b write=x01xb	
DIR4	Directory Register 4	FBh			
BCR1	BIOS Core to Bus Clock Ratio	48h		0100b	
BCR1	BIOS PLL Hot Reset	49h			
LCR1	L2_CNTL	41h			
TWR0	Table Walk 0	20h		0001b	

Note: Registers and MAPEN values not mentioned in this section are reserved.

2.5.4 Cyrix Configuration Control Registers (CCR0-CCR7)

The Configuration Control Registers (CCR0-CCR7) are used to assign non-cached memory areas, set up SMM, provide CPU identification information and control various features.

CCR1, CCR3, and CCR6 may be written at any time unless the SMI_LOCK (CCR3[0]) is set or an SMI is active.

2.5.4.1 Configuration Control Register 0 (CCR0)

7	6	5	4	3	2	1	0
Reserved						NC1	*Reserved*

Index: C0h
Default Value: 02h
Access: Read/Write
MAPEN: xxxxh

Bit	Name	Description
1	NC1	No Cache 640 KB - 1 MB 1 = Address region 640 KB to 1 MB is non-cacheable. 0 = Address region 640 KB to 1 MB is cacheable.

2.5.4.2 Configuration Control Register 1 (CCR1)

7	6	5	4	3	2	1	0
SM3	*Reserved*						

Index: C1h
Default Value: 20h
Access: Read/Write
MAPEN: xxxxh

Bit	Name	Description
7	SM3	SMM Address Space Address Region 3: 1 = Address Region 3 is designated as SMM address space. 0 = Address Region 3 is system memory.
1	Reserved	Read only (USE_SMI). Set to 1.

This register may be written at any time unless the SMI_LOCK (CCR3[0]) is set or an SMI is active.

2.5.4.3 Configuration Control Register 2 (CCR2)

7	6	5	4	3	2	1	0
Reserved			WPR1	SUSP_HLT	LOCK_NW	*Reserved*	

Index: C2h
Default Value: 00h
Access: Read/Write
MAPEN: xxxxh

Bit	Name	Description
4	WPR1	Write-Protect Region 1 1 = Designates any cacheable accesses in 640 KB to 1 MB address region are write protected.
3	SUSP_HLT	Suspend on Halt: 1 = Execution of the HLT instruction causes the CPU to enter low power suspend mode. 0 = Halt behaves normally.
2	LOCK_NW	Lock NW: 1 = NW bit (CR0[29]) becomes read-only and the CPU ignores any writes to the NW bit. 0 = NW bit (CR0[29]) can be modified.

2.5.4.4 Configuration Control Register 3 (CCR3)

7	6	5	4	3	2	1	0
MAPEN[3-0]				*Reserved*		NMI_EN	SMI_LOCK

Index: C3h
Default Value: 00h
Access: Read/Write
MAPEN: xxxxh

Bit	Name	Description
7:4	MAPEN[3-0]	**Map Enable Bits** **These four bits enable different combinations of configuration registers.** Refer to Section 2.5.2 "Map Enable (MAPEN)" on page 47. 0001 = All configuration registers are accessible. 0000 = Only configuration register with indexes: C0 - CFh, FEh, and FFh are accessible
1	NMI_EN	NMI Enable: 1 = NMI interrupt is recognized while servicing an SMI interrupt. NMI_EN should be set only while in SMM after the appropriate SMI interrupt service routine has been set up. 0 = NMI disabled while servicing SMI.
0	SMI_LOCK	SMI Lock: 1 = The following SMM configuration bits can only be modified while in an SMI service routine: CCR1: USE_SMI, SMAC, SM3 CCR3: NMI_EN CCR6: N, SMM_MODE ARR3: Starting address and block size. Once set, the features locked by SMI_LOCK cannot be unlocked until the RESET pin is asserted.

This register contains MAPEN which is a pointer to the group of registers. See MAPEN Section 2.5.2 on page 2-47.

This register may be written at any time unless the SMI_LOCK is set or an SMI is active.

2.5.4.5 Configuration Control Register 4 (CCR4)

7	6	5	4	3	2	1	0
CPUID	Reserved						

Index: E8h
Default Value: 84h
Access: Read/Write
MAPEN: 0001

Bit	Name	Description
7	CPUID	Enable CPUID Instructions: 1 = The ID bit in the FFLAGS register can be modified and execution of the CPUID instruction occurs. 0 = The ID bit in the EFLAGS register can not be modified and execution of the CPUID instruction causes an invalid opcode exception.

2.5.4.6 Configuration Control Register 5 (CCR5)

7	6	5	4	3	2	1	0
Reserved		ARREN	*Reserved*				

Index: E9h
Default Value: 00h
Access: Read/Write
MAPEN: 0001

Bit	Name	Description
5	ARREN	**Address Region Registers Enable:** Enables address decoding of ARR0-ARRD. 1 = Enables all ARR registers. 0 = Disables all ARR registers. If SM3 is set ARR3 is enabled regardless of the setting of ARREN. (SM3 is bit 7 in CCR1.)

2.5.4.7 Configuration Control Register 6 (CCR6)

7	6	5	4	3	2	1	0
Reserved							SMM_MODE

Index: EAh:
Default Value: 40h
Access: Read/Write
MAPEN: 0001

Bit	Name	Description
0	SMM_MODE	SMM Mode: 1 = Enable Cyrix-enhanced SMM mode. 0 = Disable Cyrix-enhanced SMM mode.

This register may be written at any time unless the SMI_LOCK (CCR3[0]) is set or the processor is in SMM.

2.5.4.8 Configuration Control Register 7 (CCR7)

7	6	5	4	3	2	1	0
Reserved	*Reserved*	*Reserved*	3DNOW_EN	*Reserved*	*Reserved*	*Reserved*	*Reserved*

Index: EBh
Default Value: 00h
Access: Read/Write
MAPEN: 0001

Bit	Name	Description
4	3DNOW_EN	1 = Enable 3DNOW instructions. 0 = Disable 3DNOW instructions.

2.5.4.9 Table Walk Register 0 (TWR0)

7	6	5	4	3	2	1	0
Reserved	*Reserved*	*Reserved*	Cache_TE	*Reserved*	*Reserved*	*Reserved*	*Reserved*

Index: 20h
Default Value: 00h
Access: Read/Write
MAPEN: 0001

Bit	Name	Description
4	Cache_TE	1 = Enable caching of table entries. Improves performance. 0 = Disable caching of table entries.

2.5.4.10 Address Regions in Memory

Selected regions of main memory space can be assigned different attributes. These regions are called address regions. Each address region is defined by a pair of registers—an Address Region Register (ARRn) and a Region Control Register (RCRn).

The ARRn registers are used to specify the location and size for these regions.

The RCRn registers are used to specify the attributes for these regions.

The number (n) is a hexadecimal number that designates the region number.

2.5.4.11 Address Region Registers (ARRn)

23 16	15 8	7 4	3 0
MAIN MEMORY BASE ADDRESS			SIZE

Index: See Table 2-23
Default Value: 00h
Access: Read/Write
MAPEN: See next page.

Bit	Name	Description
23-4	MAIN MEMORY BASE ADDRESS	Starting address for the particular address region. Memory address bits A[31-24] defined by ARRn bits 23 - 16 Memory address bits A[23-16] defined by ARRn bits 15 - 8 Memory address bits A[15-12] defined by ARRn bits 7 - 4
3-0	SIZE	Size of the particular address region as defined by Table 2-24 (Page 2-62).

Three I/O 22h/23h port passes are required access one of 24-bit ARRn registers. The three index numbers for all the ARRn registers are listed in Table 2-23.

Table 2-23. ARRn Register Index Assignment

ARRn Name	MAIN MEMORY BASE ADDRESS			SIZE	MAPEN CCR3[7-4]
	A31-A24	A23-A16	A15-A12		
ARR0	C4h	C5h	C6h[7-4]	C6h[3-0]	xxxxb
ARR1	C7h	C8h	C9h[7-4]	C9h[3-0]	
ARR2	CAh	CBh	CCh[7-4]	CCh[3-0]	
ARR3	CDh	CEh	CFh[7-4]	CFh[3-0]	
ARR4	D0h	D1h	D2h[7-4]	D2h[3-0]	0001b
ARR5	D3h	D4h	D5h[7-4]	D5h[3-0]	
ARR6	D6h	D7h	D8h[7-4]	D4h[3-0]	
ARR7	D9h	DAh	DBh[7-4]	DBh[3-0]	
ARR8	A4h	A5h	A6h[7-4]	A6h[3-0]	read= x010b write=x01xb
ARR9	A7h	A8h	A9h[7-4]	A9h[3-0]	
ARRA	AAh	ABh	ACh[7-4]	ACh[3-0]	
ARRB	ADh	AEh	AFh[7-4]	AFh[3-0]	
ARRC	D0h	D1h	D2h[7-4]	D2h[3-0]	
ARRD	D3h	D4h	D5h[7-4]	D5h[3-0]	

Address region 7 defines total system memory unless superseded by other address region definitions. The SIZE field is defined in two different way as listed in Table 2-24. If the address region size field is zero, the address region is disabled. After a reset, all ARR Registers are initialized to 00h. The base address of the ARR address region, selected by the Base Address field, must be on a block size boundary. For example, a 128KB block is allowed to have a starting address of 0KB, 128KB, 256KB, and so on. A 512KB block is allowed to have a starting address of 0KB, 512KB, 1024KB, and so on. Address region 3 defines SMM space when enabled by CCR1 bit 7.

Table 2-24. ARRn Address Region Size Field Definitions

Size	Block Size		
	ARR0-ARR6	ARR7-ARRB	ARRC-ARRD
00h	Disable	Disable	Disable
01h	4 KB	256 KB	256KB
02h	8 KB	512 KB	Reserved
03h	16 KB	1 MB	
04h	32 KB	2 MB	
05h	64 KB	4 MB	
06h	128 KB	8 MB	
07h	256 KB	16 MB	
08h	512 KB	32 MB	
09h	1 MB	64 MB	
0Ah	2 MB	128 MB	
0Bh	4 MB	256 MB	
0Ch	8 MB	512 MB	
0Dh	16 MB	1 GB	
0Eh	32 MB	2 GB	
0Fh	4 GB	4 GB	

2.5.4.12 Region Control Registers

7	6	5	4	3	2	1	0
Reserved	INV_RGN	WP	WT	WG	*Reserved*		RCD/RDE

*Note: RCD is defined for RCR0-RCR6. RCE is defined for RCR7.

BIT POSITION	NAME	DESCRIPTION
6	INV_RGN	ARR0 Invert Region 1 = applies the controls specified in RCRn to all memory addresses outside the region specified in ARR0.
5	WP	Write-Protect 1 = enables write protect for address region n.
4	WT	Write-Through 1 = defines the address region as write through instead of write-back. Any system ROM that is allowed to be cached by the processor should be defined as write through.
3	WG	Write-Gathering 1 = enables write gathering for address region n. With WG enabled, multiple byte, word or dword writes to sequential addresses that would normally occur as individual cycles on the bus are collapsed, or “gathered” within the processor and then completed as a single write cycle. WG improves bus utilization and should be used on memory regions that are not sensitive to gathering.
0	RCD	Cache Disable (RCR0 - RCR6 only) 1 = defines the address region n as non-cacheable.
0	RCE	Cache Enable (RCR7 only) 1 = enables caching of all memory outside of con-cachable regions

The Region Control Registers (RCRn) specify the attributes for the memory address regions defined by the corresponding Address Region Registers (ARRn). Cacheability, weak locking, write gathering and cache write-through policies can be enabled or disabled using the RCRn registers. Table 2-25 describes the index and MAPEN assignments for RCRn.

Undefined Memory Regions

If an address is accessed that is not in a memory region defined by an ARR/RCR register pair, the following conditions apply:

- If the memory address is cached, write-back is enabled
- Writes are not gathered

Overlapping Regions

If two regions specified by ARR Registers overlap and conflicting attributes are specified, the following attributes take precedence:

- Write-back is disabled
- Writes are not gathered
- Strong locking takes place
- The overlapping regions are non-cacheable

Table 2-25. RCRn Register Index Assignment

RCRn Name	Index	MAPEN CCR3[7-4]
RCR0	DCh	x00xb
RCR1	DDh	
RCR2	DEh	
RCR3	DFh	
RCR4	E0h	
RCR5	E1h	
RCR6	E2h	
RCR7	E3h	
RCR8	DCh	read= x010b write=x01xb
RCR9	DDh	
RCRA	DEh	
RCRB	DFh	
RCRC	E0h	
RCRD	E1h	

Inverted Region (INV_RGN)

Setting INV_RGN applies the controls in RCRx to all the memory addresses outside the specified address region ARRx. This bit affects RCR0-RCR6, but not RCR7

Write-Through (WT)

Setting WT defines the address region as write-through instead of write-back, assuming the region is cacheable. Regions where system ROM are loaded (shadowed or not) should be defined as write-through.

Write Gathering (WG)

Setting WG enables write gathering for the associated address region. Write gathering allows multiple byte, word, or DWORD sequential address writes to accumulate in the on-chip write buffer. As instructions are exe-

cuted the results are placed in a series of output buffers. These buffers are gathered into the final output buffer.

- When the 32-byte buffer becomes full, the contents of the buffer are written on the external 64-bit data bus. Performance is enhanced by avoiding many memory write cycles.
- WG should not be used on memory regions that are sensitive to write cycle gathering. WG can be enabled for both cacheable and non-cacheable regions.

Write Protect (WP)

- Setting WP enables write protect for the corresponding address region. With WP enabled, The memory region is treated as read-only when cached into the cpu cache. During a cache-hit write, the cache will not be modified. The data will still be written through to main memory however, and it is up to the chipset memory controller to ignore the main memory write if necessary.

Cache Disable (CD)

- Cache Disable, if set, defines the address region as non-cacheable. This bit works in conjunction with the *CR0_CD and PCD bits* to determine line cacheability. Whenever possible, the ARR/RCR combination should be used to define non-cacheable regions.

For RCR0 through RCR6

- Only one RCD, WG, WT. WP bit of each RCR register can be SET at a time.
- To define a region to be WC, define corresponding ARR6-0 and SET only WG bit in the corresponding RCR6-0 to 1.
- To define a region to be WP, define corresponding RCR6-0 and SET only WP bit in the corresponding RCR6-0 to 1.

ARR/RCR Programming Example

The following example illustrates the values used to program ARR/RCR registers. In this example, ARR4 is available. Index D0h is the most significant byte of this register; Index D2h is the least significant byte of this register. Values are programmed via the Port 22h/23h mechanism.

Scenario:

- Define a region from 31MB-32MB of memory as non-cacheable and not located in physical memory space:

1) ARR4 (MAPEN = 0001, Index D0h)= 01h	Sets AD31-AD24 to 01h
2) ARR4 (MAPEN = 0001, Index D1h)= F0h	Sets AD23-AD16 to F0h
3) ARR4 (MAPEN = 0001, Index D2h)= 09h	Sets AD15-AD12 to 0h and Size = 1MB (01F0000h = 31M)
4) RCR4 (MAPEN = 0001, Index E0h= 01h	Sets Memory Region to Non-Cacheable

2.5.5 Directory Registers

The DIR Registers allow BIOS and other software to identify the specific CPU and stepping. System Management Mode (SMM) control information is stored in the SMM registers. DIR0 and DIR1 registers are accessed by writing to the I/O Port 22h index using indices FEh and FFh. They can be read from any MAPEN except 4.

2.5.5.1 Directory Register 0 (DIR0)

7	6	5	4	3	2	1	0
Cyrix Processor Family = 8h				CPU Clock Multiplier			

Index: FEh
Default Value: Dynamic - Revision Dependent
Access: Read Only
MAPEN

Bit	Name	Description
7-4	Cyrix Processor Family	Cyrix Processor family code. Read only value set to 8.
3-0	CLK_MULT	Clock Multiplier Indicates the clock ratio. The value of CLK_MULT is set by the input pins NMI, INTR, A20M#, IGNNE#; or by BIOS through BCR1 and BCR2

TYPE Bits	Clock Ratio
4h	2.5
1h	3.0
5h	3.5
2h	4.0
6h	4.5
3h	5.0
7h	5.5
8h	6.0
Ah	6.5
9h	7.0
Bh	7.5

2.5.5.2 Directory Register 1 (DIR1)

7	6	5	4	3	2	1	0
STEP_ID				REV_ID			

Index: FFh
Default Value: Dynamic - Revision Dependent
Access: Read Only
MAPEN: xxxh

Bit	Name	Description
7-4	STEP_ID	Step Identification Indicates the major revision of the Cyrix III. This value is zero based and is usually only incremented on production revision parts.
3-0	REV_ID	Revision Identification Indicates the minor revision of the Cyrix III. This value is zero based. For example, the production revision 1.1 of a CPU is listed as 01h. This value is incremented on production revision parts.

DIR2 is reserved.

2.5.5.3 Directory Register 3 (DIR3)

7	6	5	4	3	2	1	0
TYPE				FAMILY			

Index: FCh
Default Value: Dynamic - Revision Dependent
Access: Read Only
MAPEN: Read =xxxxb, Write = x00xb

Bit	Name	Description
7 - 4	TYPE	Processor type as read by CPU_ID instruction. Read only value set to 0.
3 - 0	FAMILY	Processor family as read by CPU_ID instruction. Read only value set to 6.

2.5.5.4 Directory Register 4 (DIR0)

7	6	5	4	3	2	1	0
STEP_ID				REV_ID			

Index: FBh
Default Value: Dynamic - Revision Dependent
Access: Read Only
MAPEN: Read =xxxxb, Write = x00xb

Bit	Name	Description
7-4	MODEL	Processor model as read by CPU_ID instruction. Read only value set to 5.
3-0	STEPPING	Processor stepping as read by CPU_ID instruction.

2.5.5.5 BIOS Core-to-Bus Clock Ratio Configuration Register

Index: 48h
Default Value: 00h
Access: Read/Write
MAPEN: 0100b

Bit	Name	Description
7	HOTRST_TRIGGERED	A read/write bit used to indicate to the BIOS whether hot reset has been triggered or not.
6	Reserved.	0
5:4	BSEL[1- 0]	Indicate the P6 bus speed.
3:0	BIOS_CLKRATIO[3-0]	Core-to-bus clock ratio as described below.

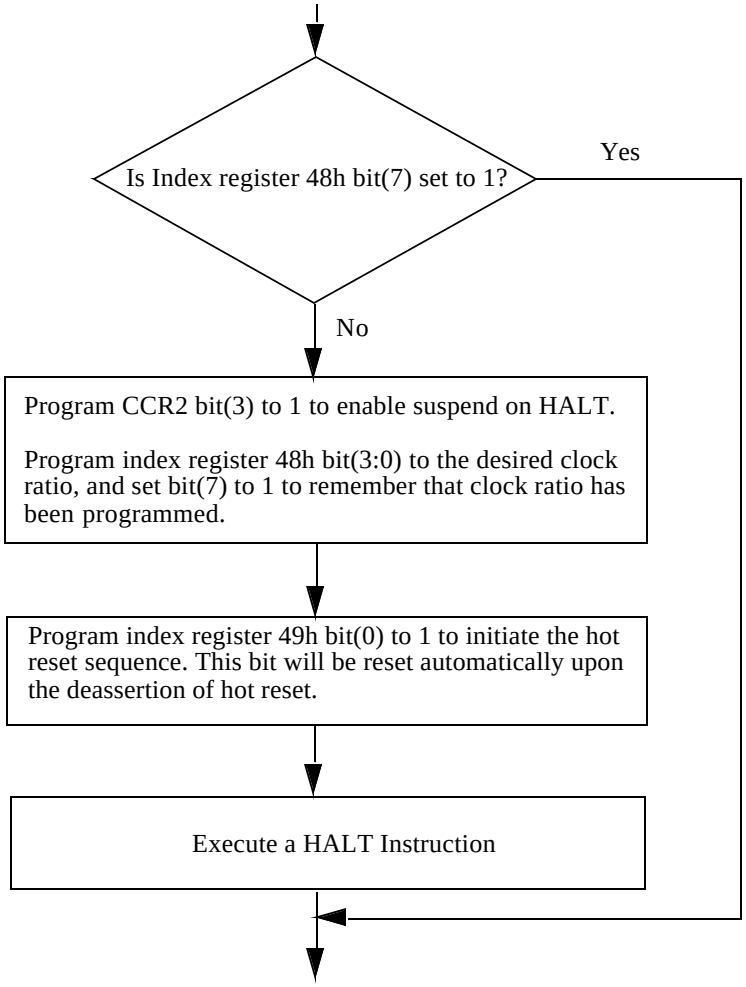
TYPE Bits	Clock Ratio
4h	2.5
1h	3.0
5h	3.5
2h	4.0
6h	4.5
3h	5.0
7h	5.5
8h	6.0
Ah	6.5
9h	7.0
Bh	7.5

Steps to set clock multiplier through BIOS

2.5.5.6 BIOS PLL Hot Reset Configuration Register

Index: 49h
Default Value: 00h
Access: Read/Write
MAPEN: 010

Bit	Name	Description
7:1	Reserved.	0
0	BIOS_HOTRESET	Writing a 1 to this bit will start the internal reset sequence to the PLL and load the BIOS_CLKRATIO[3:0] value to the PLL.



2.5.5.7 L2_CNTL Configuration Register

Index: 41h
Default Value:
Access: Read/Write
MAPEN: 0100b

Bit	Name	Description
3	L2_WT	L2 Write Through. All L1 evictions (cache line writes) are not stored in the L2. If the evicted cache line is modified, the cache line is written to the P6 bus. If the cache line is in the shared or exclusive state, it is discarded.
2	L2_ENABLE	L2 Enable. All L2 accesses (reads, writes or snoops) will miss the L2. An eviction from the L1 (cache line write) will not update the L2. A WBINV instruction must be generated when changing this bit from a 1 to a 0. The POR value of this bit is 0.

2.6 Debug Registers

Six debug registers (DR0-DR3, DR6 and DR7), shown in Figure 2-9, support debugging on the Cyrix III CPU. The bit definitions for the debug registers are listed in Table 2-26 (Page 2-74).

Memory addresses loaded in the debug registers, referred to as “breakpoints”, generate a debug exception when a memory access of the specified type occurs to the specified address. A data breakpoint can be specified for a particular kind of memory access such as a read or a write. Code breakpoints can also be set allowing debug exceptions to occur whenever a given code access (execution) occurs.

The size of the debug target can be set to 1, 2, or 4bytes. The debug registers are accessed via MOV instructions which can be executed only at privilege level 0.

The Debug Address Registers (DR0-DR3) each contain the linear address for one of four possible breakpoints. Each breakpoint is further specified by bits in the Debug Control Register (DR7). For each breakpoint address in DR0-DR3, there are corresponding fields L, R/W, and LEN in DR7 that specify the type of memory access associated with the breakpoint.

The R/W field can be used to specify instruction execution as well as data access breakpoints. Instruction execution breakpoints are always taken before execution of the instruction that matches the breakpoint.

The Debug Status Register (DR6) reflects conditions that were in effect at the time the debug exception occurred. The contents of the DR6 register are not automatically cleared by the processor after a debug exception occurs and, therefore, should be cleared by software at the appropriate time.

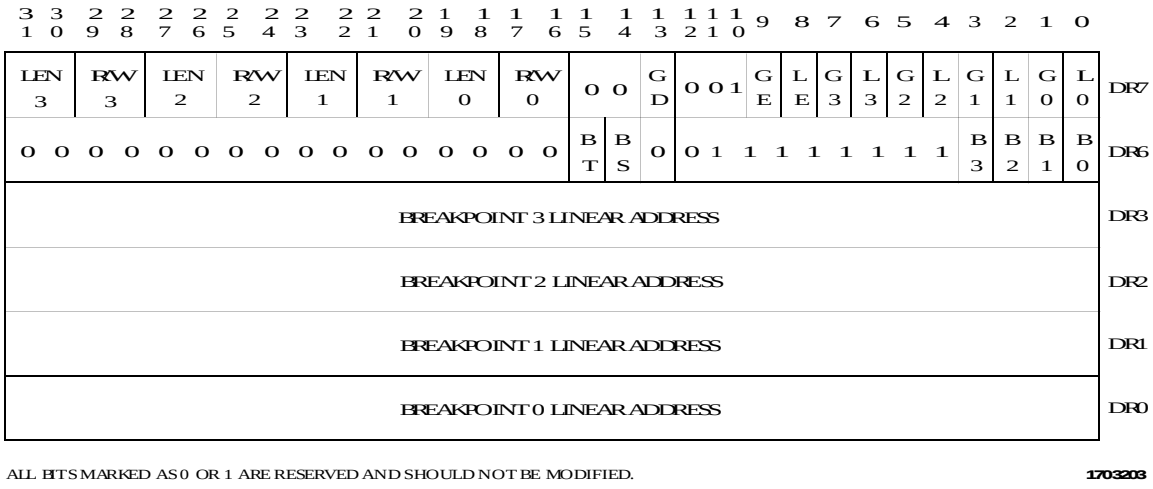


Figure 2-9. Debug Registers

Code execution breakpoints may also be generated by placing the breakpoint instruction (INT 3) at the location where control is to be regained. Additionally, the single-step feature may be enabled by setting the TF flag in the EFLAGS register. This causes the processor to perform a debug exception after the execution of every instruction.

Table 2-26. DR6 and DR7 Debug Register Field Definitions

REGISTER	FIELD	NUMBER OF BITS	DESCRIPTION
DR6	BI	1	BI is set by the processor if the conditions described by DRI, R/Wi, and LENI occurred when the debug exception occurred, even if the breakpoint is not enabled via the GI or LI bits.
	BT	1	BT is set by the processor before entering the debug handler if a task switch has occurred to a task with the T bit in the TSS set.
	BS	1	BS is set by the processor if the debug exception was triggered by the single-step execution mode (TF flag in EFLAGS set).
DR7	R/Wi	2	Specifies type of break for the linear address in DR0, DR1, DR3, DR4: 00 - Break on instruction execution only 01 - Break on data writes only 10 - Not used 11 - Break on data reads or writes.
	LENI	2	Specifies length of the linear address in DR0, DR1, DR3, DR4: 00 - One byte length 01 - Two byte length 10 - Not used 11 - Four byte length.
	Gi	1	If set to a 1, breakpoint in DRI is globally enabled for all tasks and is not cleared by the processor as the result of a task switch.
	LI	1	If set to a 1, breakpoint in DRI is locally enabled for the current task and is cleared by the processor as the result of a task switch.
	GD	1	Global disable of debug register access. GD bit is cleared whenever a debug exception occurs.

2.7 Address Space

The Cyrix III CPU can directly address 64 KB of I/O space and 4 GB of physical memory (Figure 2-24).

Memory Address Space. Access can be made to memory addresses between 0000 0000h and FFFF FFFFh. This 4 GB memory space can be accessed using byte, word (16 bits), or doubleword (32 bits) format. Words and doublewords are stored in consecutive memory bytes with the low-order byte located in the lowest address. The physical address of a word or doubleword is the byte address of the low-order byte.

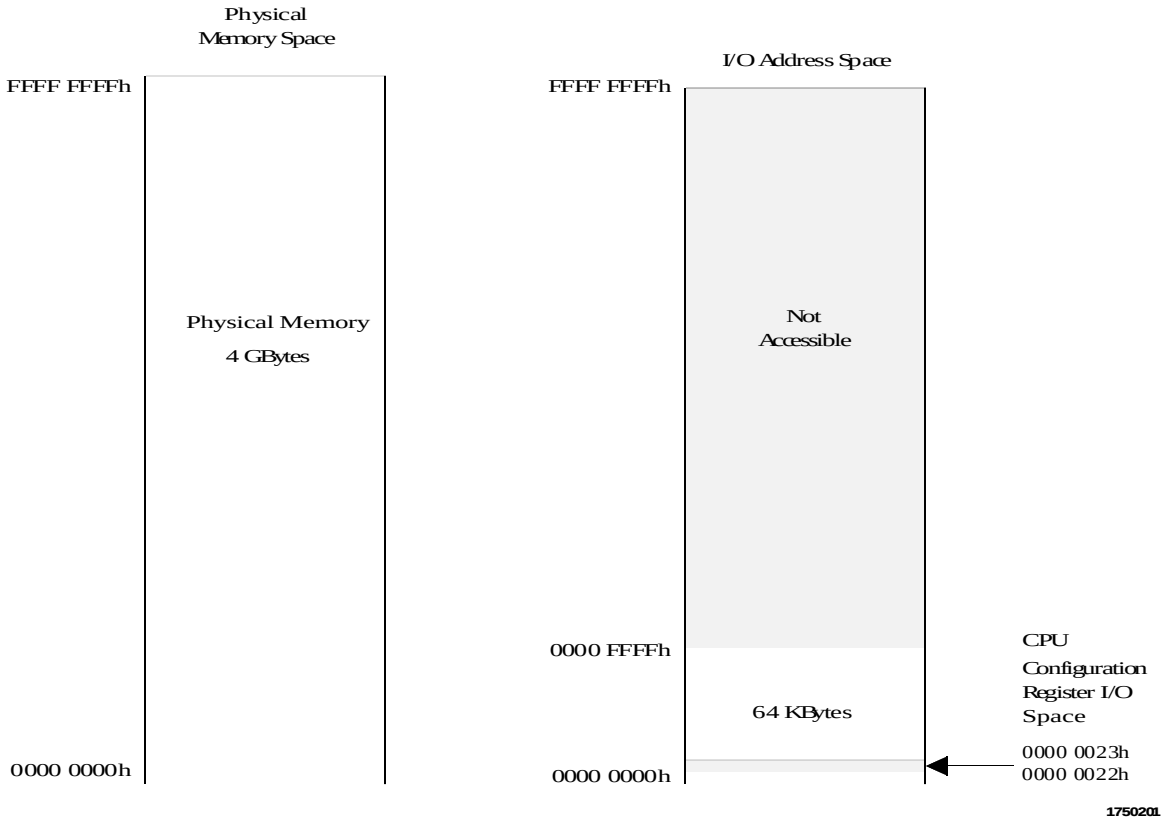


Figure 2-24. Memory and I/O Address Spaces

I/O Address Space

The Cyrix III I/O address space is accessed using IN and OUT instructions to addresses referred to as “ports”. The accessible I/O address space size is 64 KB and can be accessed through 8-bit, 16-bit or 32-bit ports.

The accessible I/O address space ranges between locations 0000 0000h and 0000FFFFh (64 KB). The I/O locations (ports) 22h and 23h can be used to access the Cyrix III configuration registers.

2.8 Memory Addressing Methods

With the Cyrix III CPU, memory can be addressed using nine different addressing modes (Table 2-26, Page 2-77). These addressing modes are used to calculate an offset address often referred to as an effective address. Depending on the operating mode of the CPU, the offset is then combined using memory management mechanisms to create a physical address that actually addresses the physical memory devices.

Memory management mechanisms on the Cyrix III CPU consist of segmentation and paging. Segmentation allows each program to use several independent, protected address spaces. Paging supports a memory subsystem that simulates a large address space using a small amount of RAM and disk storage for physical memory. Either or both of these mechanisms can be used for management of the Cyrix III CPU memory address space.

2.8.1 Offset Mechanism

The offset mechanism computes an offset (effective) address by adding together one or more of three values: a base, an index and a displacement. When present, the base is the value of one of the eight 32-bit general registers. The index if present, like the base, is a value that is in one of the eight 32-bit general purpose registers (not including the ESP register). The index differs from the base in that the index is first multiplied by a scale factor of 1, 2, 4 or 8 before the summation is made. The third component added to the memory address calculation is the displacement. The displacement is a value of up to 32-bits in length supplied as part of the instruction. Figure 2-25 illustrates the calculation of the offset address.

Nine valid combinations of the base, index, scale factor and displacement can be used with the Cyrix III CPU instruction set. These combinations are listed in Table 2-26. The base and index both refer to contents of a register as indicated by [Base] and [Index].

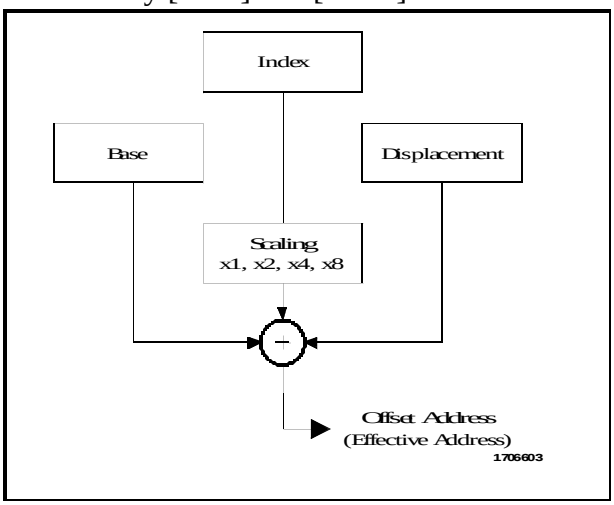


Figure 2-25. Offset Address Calculation

Table 2-26. Memory Addressing Modes

ADDRESSING MODE	BASE	INDEX	SCALE FACTOR (SF)	DISPLACEMENT (DP)	OFFSET ADDRESS (OA) CALCULATION
Direct				x	OA = DP
Register Indirect	x				OA = [BASE]
Based	x			x	OA = [BASE] + DP
Index		x		x	OA = [INDEX] + DP
Scaled Index		x	x	x	OA = ([INDEX] * SF) + DP
Based Index	x	x			OA = [BASE] + [INDEX]
Based Scaled Index	x	x	x		OA = [BASE] + ([INDEX] * SF)
Based Index with Displacement	x	x		x	OA = [BASE] + [INDEX] + DP
Based Scaled Index with Displacement	x	x	x	x	OA = [BASE] + ([INDEX] * SF) + DP

2.8.2 Memory Addressing

Real Mode Memory Addressing

In real mode operation, the Cyrix III CPU only addresses the lowest 1 MB of memory. To calculate a physical memory address, the 16-bit segment base address located in the selected segment register is multiplied by 16 and then the 16-bit offset address is added. The resulting 20-bit address is then extended. Three hexadecimal zeros are added as upper address bits to create the 32-bit physical address. Figure 2-26 illustrates the real mode address calculation.

The addition of the base address and the offset address may result in a carry. Therefore, the resulting address may actually contain up to 21 significant address bits that can address memory in the first 64 KB above 1 MB.

Protected Mode Memory Addressing

In protected mode three mechanisms calculate a physical memory address (Figure 2-27, Page 2-79).

- Offset Mechanism that produces the offset or effective address as in real mode.
- Selector Mechanism that produces the base address.
- Optional Paging Mechanism that translates a linear address to the physical memory address.

The offset and base address are added together to produce the linear address. If paging is not enabled, the linear address is used as the physical memory address. If paging is enabled, the paging mechanism is used to translate the linear address into the physical address. The offset mechanism is described earlier in this section and applies to both real and protected mode. The selector and paging mechanisms are described in the following paragraphs.

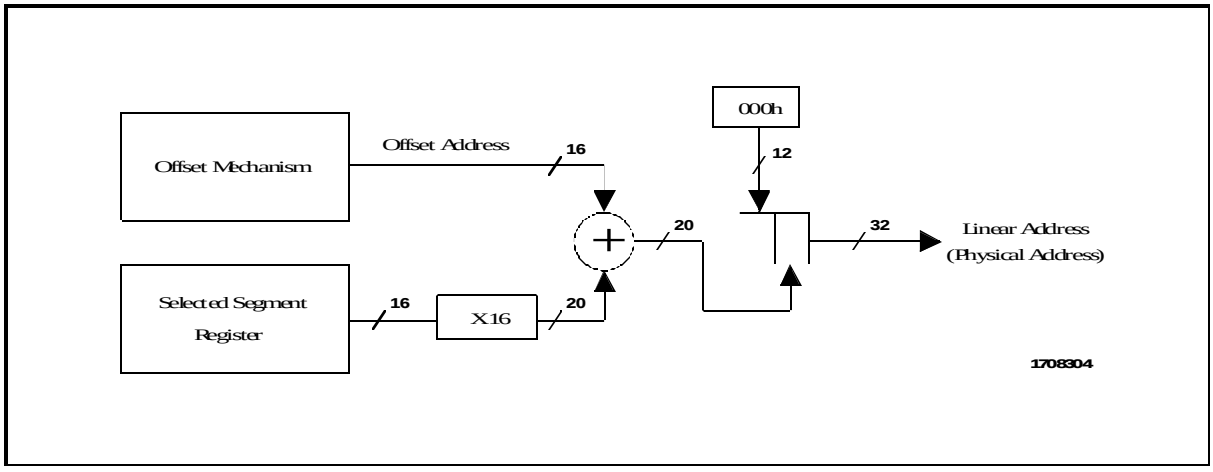


Figure 2-26. Real Mode Address Calculation

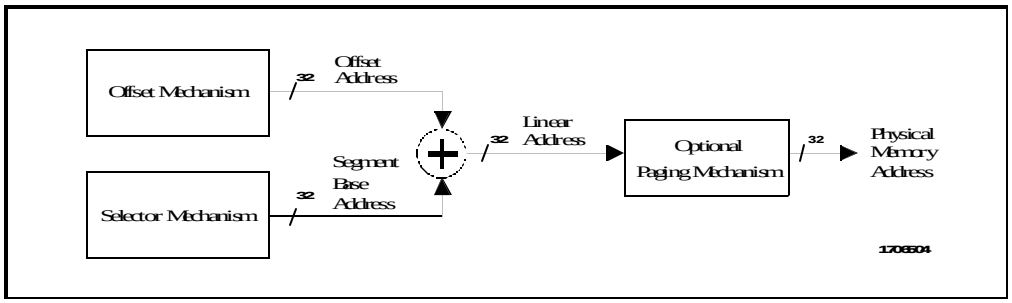


Figure 2-27. Protected Mode Address Calculation

2.8.3 Selector Mechanism

Using segmentation, memory is divided into an arbitrary number of segments, each containing usually much less than the 2^{32} byte (4 GB) maximum.

The six segment registers (CS, DS, SS, ES, FS and GS) each contain a 16-bit selector that is used when the register is loaded to locate a segment descriptor in either the global descriptor table (GDT) or the local descriptor table (LDT). The segment descriptor defines the base address, limit, and attributes of the selected segment and is cached on the Cyrix III CPU as a result of loading the selector. The cached descriptor contents are not visible to the programmer. When a memory reference occurs in protected mode, the linear address is generated by adding the segment base address in the hidden portion of the segment register to the offset address. If paging is not enabled, this linear address is used as the physical memory address. Figure 2-28 illustrates the operation of the selector mechanism.

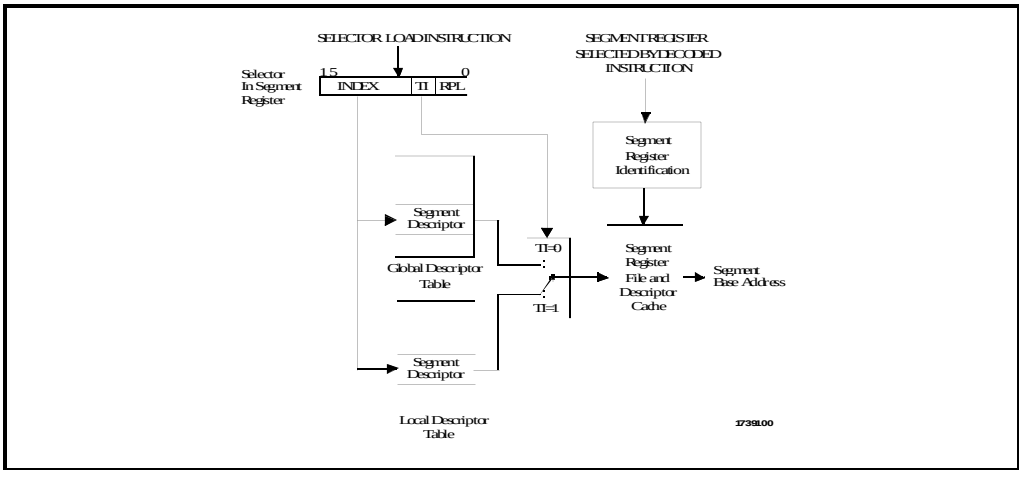


Figure 2-28. Selector Mechanism

2.8.4 Paging Mechanism

The paging mechanism translates linear addresses to their corresponding physical addresses. The page size is always 4KB. Paging is activated when the PG and the PE bits within the CR0 register are set.

The paging mechanism translates the 20 most significant bits of a linear address to a physical address. The linear address is divided into three fields DTI, PTI, PFO (, Page 2-81). These fields respectively select:

- an entry in the directory table,
- an entry in the page table selected by the directory table
- the offset in the physical page selected by the page table

The directory table and all the page tables can be considered as pages as they are 4 KB in size and are aligned on 4 KB boundaries. Each entry in these tables is 32 bits in length. The fields within the entries are detailed in Figure 2-30 (Page 2-81) and Table 2-27 (Page 2-82).

A single page directory table can address up to 4 GB of virtual memory (1,024 page tables—each table can select 1,024 pages and each page contains 4KB).

Translation Lookaside Buffer (TLB) is made up of two caches (Page 2-81).

- the L1 TLB caches page tables entries
- the L2 TLB stores PTEs that have been evicted from the L1 TLB

The L1 TLB is a 16-entry direct-mapped dual ported cache. The L2 TLB is a 384 entry, 6-way, dual ported cache.

.

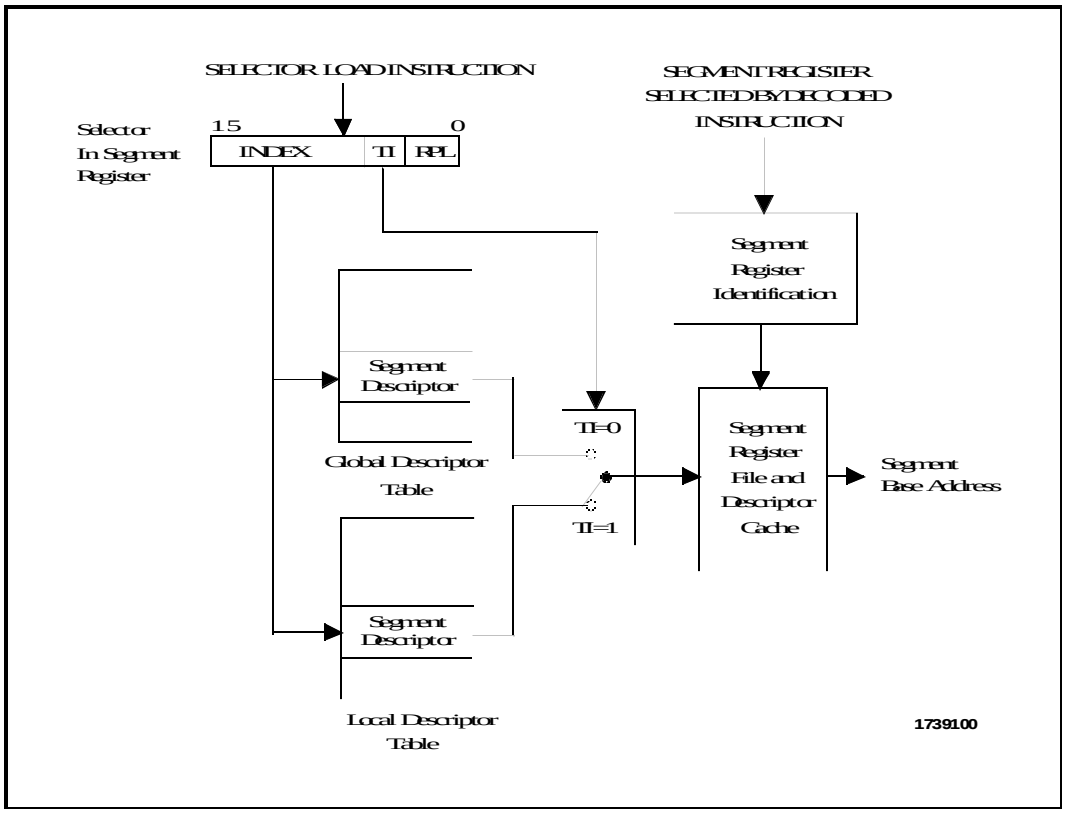


Figure 2-29. Paging Mechanism

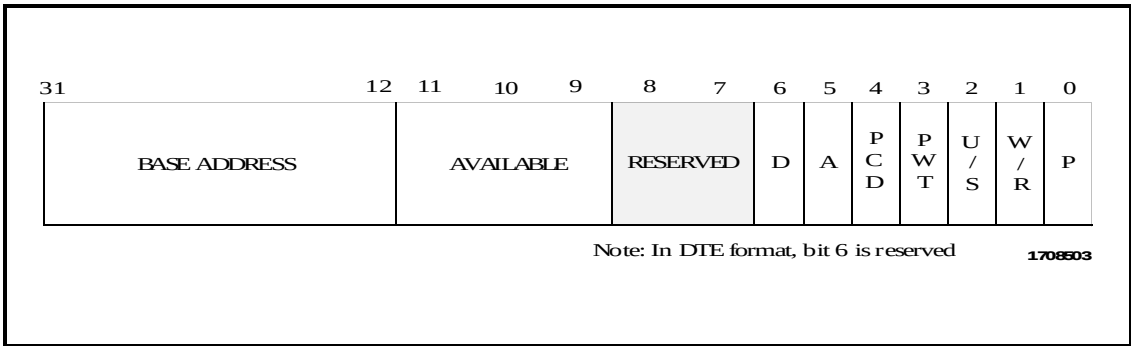


Figure 2-30. Directory and Page Table Entry (DTE and PTE) Format

Table 2-27. Directory and Page Table Entry (DTE and PTE) Bit Definitions

BIT POSITION	FIELD NAME	DESCRIPTION
31-12	BASE ADDRESS	Specifies the base address of the page or page table.
11-9	--	Undefined and available to the programmer.
8-7	--	Reserved and not available to the programmer.
6	D	Dirty Bit. If set, indicates that a write access has occurred to the page (PTE only, undefined in DTE).
5	A	Accessed Flag. If set, indicates that a read access or write access has occurred to the page.
4	PCD	Page Caching Disable Flag. If set, indicates that the page is not cacheable in the on-chip cache.
3	PWT	Page Write-Through Flag. If set, indicates that writes to the page or page tables that hit in the on-chip cache must update both the cache and external memory.
2	U/S	User/Supervisor Attribute. If set (user), page is accessible at privilege level 3. If clear (supervisor), page is accessible only when CPL ≤ 2.
1	W/R	Write/Read Attribute. If set (write), page is writable. If clear (read), page is read only.
0	P	Present Flag. If set, indicates that the page is present in RAM memory, and validates the remaining DTE/PTE bits. If clear, indicates that the page is not present in memory and the remaining DTE/PTE bits can be used by the programmer.

For a TLB hit, the TLB eliminates accesses to external directory and page tables.

The L1 TLB is a small cache optimized for speed whereas the L2 TLB is a much larger cache optimized for capacity. The L2 TLB is a proper superset of the L1 TLB.

The TLB must be flushed by the software when entries in the page tables are changed. Both the L1 and L2 TLBs are flushed whenever the CR3 register is loaded. A particular page can be flushed from the TLBs by using the INVLPG instruction.

2.8.4.1 Translation Lookaside Buffer Testing

The L1 and L2 Translation Lookaside Buffers (TLBs) can be tested by writing, then reading from the same TLB location. The operation to be performed is determined by the command (CMD) field Table 2-28 (Page 2-82) in the TR6 register.

Table 2-28. CMD Field

CMD	OPERATION	LINEAR ADDRESS BITS
x00	Write to L1	15 - 12
x01	Write to L2	17 - 12
010	Read from L1 X port	15 -12
011	Read from L2 X port	17 -12
110	Read from L1 Y port	15 - 12

Table 2-28. CMD Field

110	Read from L2 Y port	17 - 12

TLB Write

To perform a write to the Cyrix III TLBs, the TR7 register (Figure 2-31) is loaded with the desired physical address as well as the PCD and PWT bits. For a write to the L2 TLB, the SET field of TR7 must be also specified. The H1, H2, and HSET fields of TR7 are not used. The TR6 register is then loaded with the linear address, V, D, U, W and A fields and the appropriate CMD. For a L1 TLB write, the TLB entry is selected by bits 15-12 of the linear address. For a L2 TLB write, the TLB entry is selected by bits 17-12 of the linear address and the SET field of TR7.

TLB Read

For a L1 LTB read, the TR6 register is loaded with the linear address and the appropriate CMD. The L1 TLB entry selected by bits 15-12 of the linear address will then be accessed. The linear address, V, D, PG, U, W and A fields of TR6 and the physical address, PCD and PWT fields of TR7 are loaded from the specified L1 entry. The H1 bit of TR7 will indicate if the specified linear address hit in the L1 TLB.

For a L2 TLB read, the TR7 register is loaded with the desired SET. The TR6 register is then loaded with the linear address and the appropriate CMD. The L2 TLB entry selected by bits 17-12 of the linear address and the SET field in TR7 will then be accessed. The linear address, V,D, PG, V, W, and A fields of TR6 and the physical address, PCD and PWT fields of TR7 are loaded from the specified L2 entry. The H2 bit of TR7 will indicate if the specified linear address hit in the L2 TLB. If there was an L2 hit, the HSET field of TR7 will indicate which SET hit.

The TLB test register fields are defined in Table 2-29. (Page 2-84).

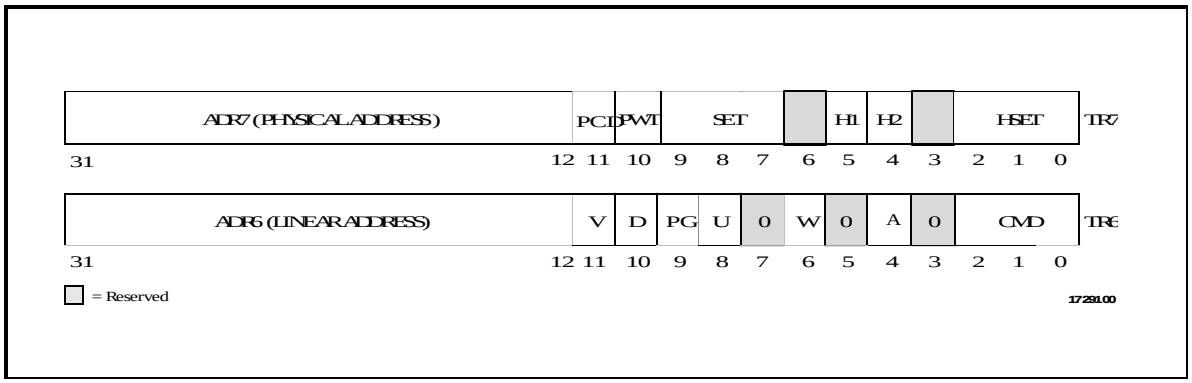


Figure 2-31. TLB Test Registers

Table 2-29. TLB Test Register Bit Definitions

REGISTER NAME	NAME	RANGE	DESCRIPTION
TR7	ADR7	31-12	Physical address or variable page size mechanism mask. TLB lookup: data field from the TLB. TLB write: data field written into the TLB.
	PCD	11	Page-level cache disable bit (PCD). Corresponds to the PCD bit of a page table entry.
	PWT	10	Page-level cache write-through bit (PWT). Corresponds to the PWT bit of a page table entry.
	SET	9-7	L2 TLB Set Selection (0h - 5h)
	H1	5	Hit in L1 TLB
	H2	4	Hit in L2 TLB
	HSET	2-0	L2 Set Selection when L2 TLB hit occurred (0h - 5h)
TR6	ADR6	31-12	Linear Address. TLB lookup: The TLB is interrogated per this address. If one and only one match occurs in the TLB, the rest of the fields in TR6 and TR7 are updated per the matching TLB entry. TLB write: A TLB entry is allocated to this linear address.
	V	11	PTE Valid. TLB write: If set, indicates that the TLB entry contains valid data. If clear, target entry is invalidated.
	D	10	Dirty Attribute Bit
	PG	9	Page Global
	U	8	User/Supervisor Attribute Bit
	W	6	Write Protect bit.
	CMD	2-0	Array Command Select. Determines TLB array command. Refer to Table 2-28 (Page 2-82).

2.9 Memory Caches

The Cyrix III CPU contains two memory caches as described in Chapter 1. The Unified Cache acts as the primary data cache, and secondary instruction cache. The Instruction Line Cache is the primary instruction cache and provides a high speed instruction stream for the Integer Unit.

The unified cache is dual-ported allowing simultaneous access to any two unique banks. Two different banks may be accessed at the same time permitting any two of the following operations to occur in parallel:

- Code fetch
- Data read (X pipe, Y pipe or FPU)
- Data write (X pipe, Y pipe or FPU).

2.9.1 Unified Cache MESI States

The unified cache lines are assigned one of four MESI states as determined by MESI bits stored in tag memory. Each 32-byte cache line is divided into two 16-byte sectors. Each sector contains its own MESI bits. The four MESI states are described below:

Modified MESI cache lines are those that have been updated by the CPU, but the corresponding main memory location has not yet been updated by an external write cycle. Modified cache lines are referred to as dirty cache lines.

Exclusive MESI lines are lines that are exclusive to the Cyrix III CPU and are not duplicated within another caching agent's cache within the same system. A write to this cache line may be performed without issuing an external write cycle.

Shared MESI lines may be present in another caching agent's cache within the same system. A write to this cache line forces a corresponding external write cycle.

Invalid MESI lines are cache lines that do not contain any valid data.

2.9.1.1 Unified Cache Testing

The TR3, TR4, and TR5 on-chip test registers provide information so the unified cache can be tested. This information determines what particular area will be tested. Fields within these test registers identify which area of the cache will be selected for testing.

Cache Organization. The unified cache (Figure 2-32) is divided into 32-bytes lines. This cache is divided into four sets. Since a set (as well as the cache) is smaller than main memory, each line in the set corresponds to more than one line in main memory. When a cache line is allocated, bits A31-A14 of the main memory address are stored in the cache line tag. The remaining address bits are used to identify the specific 32-byte cache line (A13-A5), and the specific 4-byte entry within the cache line (A4-A2).

Test Initiation. A test register operation is initiated by writing to the TR5 register shown in Figure 2-33 (Page 2-87) using a special MOV instruction. The TR5 CTL field, detailed in Table 2-30 (Page 2-87), determines the function to be performed. For cache writes, the registers TR4 and TR3 must be initialized before a write is made to TR5. Eight 4-byte accesses are required to access a complete cache line.

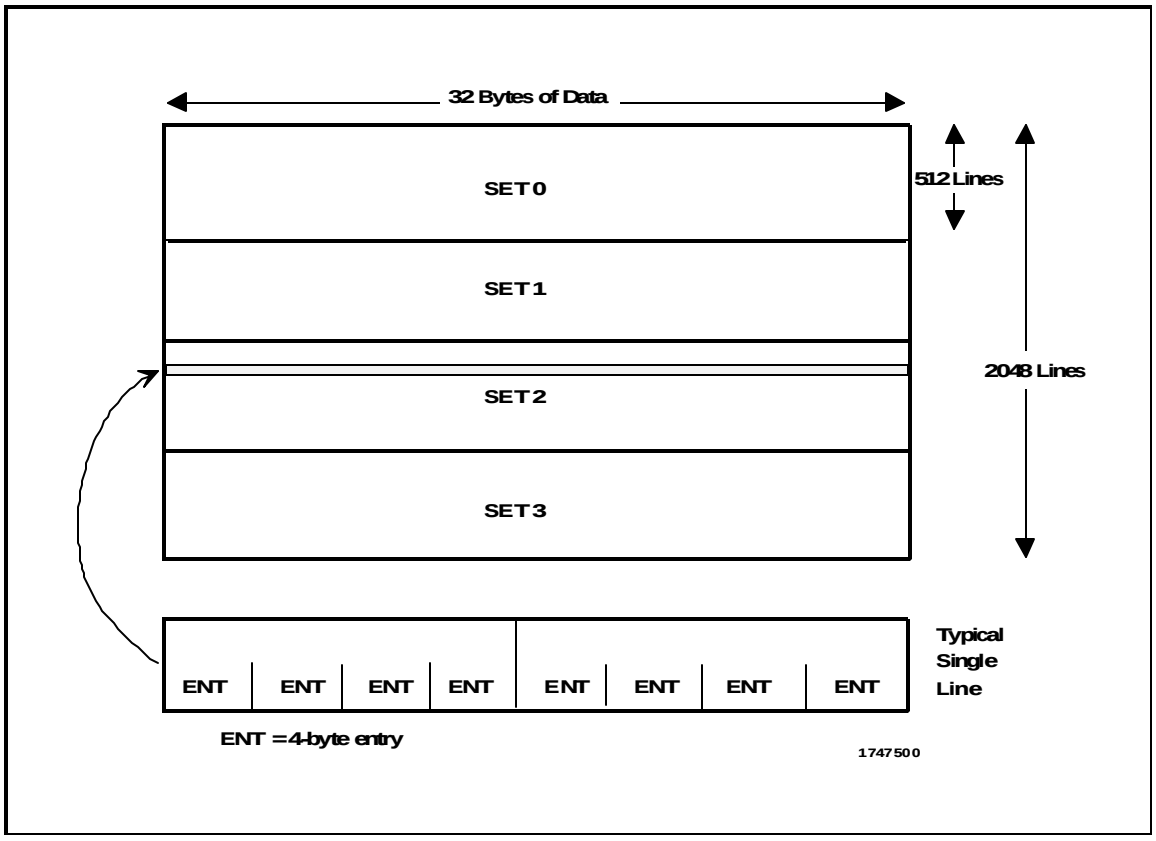


Figure 2-32. Unified Cache

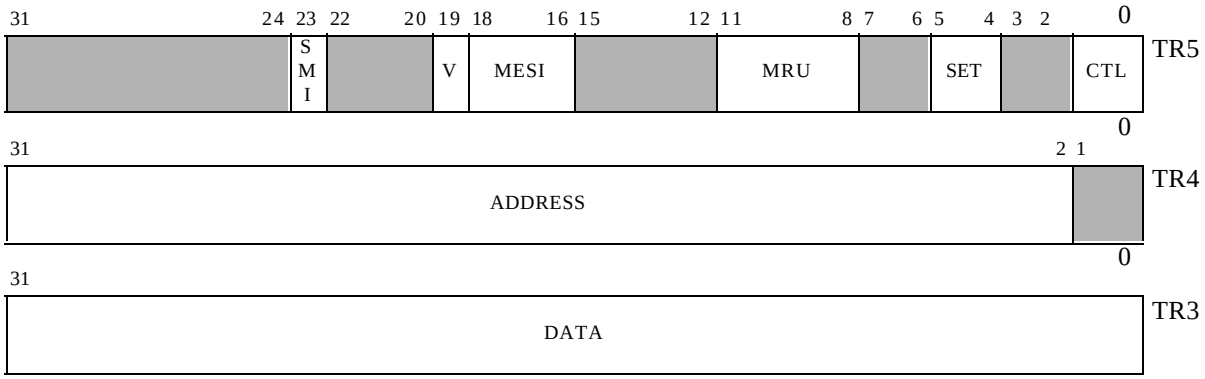


Figure 2-33. Cache Test Registers

Table 2-30. Cache Test Register Bit Definitions

REGISTER NAME	FIELD NAME	RANGE	DESCRIPTION
TR5	SMI	23	SMI Address Bit. Selects separate/cacheable SMI code/data space
	V, MESI	19 - 16	Valid, MESI Bits* If = 1000, Modified If = 1001, Shared If = 1010, Exclusive If = 0011, Invalid If = 1100, Locked Valid If = 0111, Locked Invalid Else = Undefined
	MRU	11 - 8	Used to determine the Least Recently Used (LRU) line.
	SET	5 - 4	Cache Set. Selects one of four cache sets to perform operation on.
	CTL	1 - 0	Control field If = 00: flush cache without invalidate If = 01: write cache If = 10: read cache If = 11: no cache or test register modification
TR4	ADDRESS	31 - 2	Physical Address
TR3	DATA	31 - 0	Data written or read during a cache test.

*Note: All 32 bytes should contain valid data before a line is marked as valid.

Write Operations. During a write, the TR3 DATA (32-bits) and TAG field information is written to the address selected by the ADDRESS field in TR4 and the SET field in TR5.

Read Operations. During a read, the cache address selected by the ADDRESS field in TR4 and the SET field in TR5. The TVB, MESI and MRU fields in TR5 are updated with the information from the selected line. TR3 holds the selected read data.

Cache Flushing. A cache flush occurs during a TR5 write if the CTL field is set to zero. During flushing, the CPU's cache controller reads through all the lines in the cache. "Modified" lines are redefined as "shared" by setting the shared MESI bit. Clean lines are left in their original state.

2.9.2 RAM Cache Locking

RAM cache locking (was called Scratch Pad Memory) sets up a private area of memory that can be assigned within the Cyrix III unified cache. Cached locked RAM is read/writable and is NOT kept coherent with the rest of the system. Scratch Pad Memory is a separate memory on certain Cyrix CPUs.

Cache locking may be implemented differently on different processors. On the Cyrix III CPU, the cache locking RAM may be assigned on a cache line granularity.

RDMSR and WRMSR instructions (Page 2-39) with indices 03h to 05h are used to assign scratch pad memory. These instructions access the cache test registers. See section 2.9.1.1 (Page 2-86) for detailed description of cache test register operation. The cache line is assigned into Scratch Pad RAM by setting its MESI state to “locked valid.”

When locking physical addresses into the cache (Table 2-31), the programmer should be aware of several issues:

1) Locking all sets of the cache should not be done. It is required that one set always be available for general purpose caching. 2) Care must be taken by the programmer not to create synonyms. This is done by first checking to see if a particular address is locked before attempting to lock the address. If synonyms are created, Cyrix III CPU operation will be undefined.

When ever possible, it is recommended to flush the cache before assigning locked memory areas. Locked areas of the cache are cleared on reset, and are unaffected by warm reset and FLUSH#, or the INVD and WBINVD instructions.

Table 2-31. RAM Cache Locking Operations

Read/Write	ECX	EDX	EAX	Operation
Read/Write	03h	----	Data to be read or written from/to the cache.	Loads or stores data to/from TR3.
Write	04h	----	32 bits of address	Address in EAX is loaded into TR4. This address is the cache line address that will be locked.
Read	04h	----	32 bits of address	Stores the contents of TR4 in EAX
Write	05h	----	Data to be written into TR5	Performs operation specified in CTL field of TR5.
Read	05h	----	Data in TR5 register	Reads data in TR5 and stores in EAX.

2.10 Interrupts and Exceptions

The processing of an interrupt or an exception changes the normal sequential flow of a program by transferring program control to a selected service routine. Except for SMM interrupts, the location of the selected service routine is determined by one of the interrupt vectors stored in the interrupt descriptor table.

Hardware interrupts are generated by signal sources external to the CPU. All exceptions (including so-called software interrupts) are produced internally by the CPU.

2.10.1 Interrupts

External events can interrupt normal program execution by using one of the three interrupt pins on the Cyrix III CPU.

- Non-maskable Interrupt (NMI pin)
- Maskable Interrupt (INTR pin)
- SMM Interrupt (SMI# pin).

For most interrupts, program transfer to the interrupt routine occurs after the current instruction has been completed. When the execution returns to the original program, it begins immediately following the last completed instruction.

With the exception of string operations, interrupts are acknowledged between instructions. Long string operations have interrupt windows between memory moves that allow interrupts to be acknowledged.

The NMI interrupt cannot be masked by software and always uses interrupt vector 2 to locate its service routine. Since the interrupt vector is fixed and is supplied internally, no interrupt acknowledge bus cycles are performed. This interrupt is normally reserved for unusual situations such as parity errors and has priority over INTR interrupts.

Once NMI processing has started, no additional NMIs are processed until an IRET instruction is executed, typically at the end of the NMI service routine. If NMI is re-asserted prior to execution of the IRET instruction, one and only one NMI rising edge is stored and processed after execution of the next IRET. During the NMI service routine, maskable interrupts may be enabled (unmasked). If an unmasked INTR occurs during the NMI service routine, the INTR is serviced and execution returns to the NMI service routine following the next IRET. If a HALT instruction is executed within the NMI service routine, the Cyrix III CPU restarts execution only in response to RESET#, an unmasked INTR or an SMM interrupt. NMI does not restart CPU execution under this condition.

The INTR interrupt is unmasked when the Interrupt Enable Flag (IF) in the EFLAGS register is set to 1. When an INTR interrupt occurs, the CPU performs two locked interrupt acknowledge bus cycles. During the second cycle, the CPU reads an 8-bit vector that is supplied by an external interrupt controller. This vector selects one of the 256 possible interrupt handlers which will be executed in response to the interrupt.

The SMM interrupt has higher priority than either INTR or NMI. After SMI# is asserted, program execution is passed to an SMI service routine that runs in SMM address space reserved for this purpose. The remainder of this section does not apply to the SMM interrupts. SMM interrupts are described in greater detail later in this chapter.

2.10.2 Exceptions

Exceptions are generated by an interrupt instruction or a program error. Exceptions are classified as traps, faults or aborts depending on the mechanism used to report them and the restart ability of the instruction that first caused the exception.

A Trap Exception is reported immediately following the instruction that generated the trap exception. Trap exceptions are generated by execution of a software interrupt instruction (INTO, INT 3, INT n, BOUND), by a single-step operation or by a data breakpoint.

Software interrupts can be used to simulate hardware interrupts. For example, an INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table. Execution of the interrupt service routine occurs regardless of the state of the IF flag in the EFLAGS register.

The one byte INT 3, or breakpoint interrupt (vector 3), is a particular case of the INT n instruction. By inserting this one byte instruction in a program, the user can set breakpoints in the code that can be used during debug.

Single-step operation is enabled by setting the TF bit in the EFLAGS register. When TF is set, the CPU generates a debug exception (vector 1) after the execution of every instruction. Data breakpoints also generate a debug exception and are specified by loading the debug registers (DR0-DR7) with the appropriate values.

A Fault Exception is reported prior to completion of the instruction that generated the exception. By reporting the fault prior to instruction completion, the CPU is left in a state that allows the instruction to be restarted and the effects of the faulting instruction to be nullified. Fault exceptions include divide-by-zero errors, invalid opcodes, page faults and coprocessor errors. Instruction breakpoints (vector 1) are also handled as faults. After execution of the fault service routine, the instruction pointer points to the instruction that caused the fault.

An Abort Exception is a type of fault exception that is severe enough that the CPU cannot restart the program at the faulting instruction. The double fault (vector 8) is the only abort exception that occurs on the Cyrix III CPU.

2.10.3 Interrupt Vectors

When the CPU services an interrupt or exception, the current program's FLAGS, code segment and instruction pointer are pushed onto the stack to allow resumption of execution of the interrupted program. In protected mode, the processor also saves an error code for some exceptions. Program control is then transferred to the interrupt handler (also called the interrupt service routine). Upon execution of an IRET at the end of the service routine, program execution resumes by popping from the stack, the instruction pointer, code segment, and FLAGS.

Interrupt Vector Assignments

Each interrupt (except SMI#) and exception is assigned one of 256 interrupt vector numbers Table 2-32, (Page 2-93). The first 32 interrupt vector assignments are defined or reserved. INT instructions acting as software interrupts may use any of the interrupt vectors, 0 through 255.

Table 2-32. Interrupt Vector Assignments

INTERRUPT VECTOR	FUNCTION	EXCEPTION TYPE
0	Divide error	FAULT
1	Debug exception	TRAP/FAULT*
2	NMI interrupt	
3	Breakpoint	TRAP
4	Interrupt on overflow	TRAP
5	BOUND range exceeded	FAULT
6	Invalid opcode	FAULT
7	Device not available	FAULT
8	Double fault	ABORT
9	Reserved	
10	Invalid TSS	FAULT
11	Segment not present	FAULT
12	Stack fault	FAULT
13	General protection fault	TRAP/FAULT
14	Page fault	FAULT
15	Reserved	
16	FPU error	FAULT
17	Alignment check exception	FAULT
18-31	Reserved	
32-255	Maskable hardware interrupts	TRAP
0-255	Programmed interrupt	TRAP

*Note: Data breakpoints and single-steps are traps. All other debug exceptions are faults.

In response to a maskable hardware interrupt (INTR), the Cyrix III CPU issues an interrupt acknowledge bus cycle to read the vector number from external hardware. These vectors should be in the range 32 - 255 as vectors 0 - 31 are reserved.

Interrupt Descriptor Table

The interrupt vector number is used by the Cyrix III CPU to locate an entry in the interrupt descriptor table (IDT). In real mode, each IDT entry consists of a four-byte far pointer to the beginning of the corresponding interrupt service routine. In protected mode, each IDT entry is an eight-byte descriptor. The Interrupt Descriptor Table Register (IDTR) specifies the beginning address and limit of the IDT. Following reset, the IDTR contains a base address of 0h with a limit of 3FFh.

The IDT can be located anywhere in physical memory as determined by the IDTR register. The IDT may contain different types of descriptors: interrupt gates, trap gates and task gates. Interrupt gates are used primarily to enter a hardware interrupt handler. Trap gates are generally used to enter an exception handler or software interrupt handler. If an interrupt gate is used, the Interrupt Enable Flag (IF) in the EFLAGS register is cleared before the interrupt handler is entered. Task gates are used to make the transition to a new task.

2.10.4 Interrupt and Exception Priorities

As the Cyrix III CPU executes instructions, it follows a consistent policy for prioritizing exceptions and hardware interrupts. The priorities for competing interrupts and exceptions are listed in Table 2-33 (Page 2-95). Debug traps for the previous instruction and the next instructions always take precedence. SMM interrupts are the next priority. When NMI and maskable INTR interrupts are both detected at the same instruction boundary, the Cyrix III processor services the NMI interrupt first.

The Cyrix III CPU checks for exceptions in parallel with instruction decoding and execution. Several exceptions can result from a single instruction. However, only one exception is generated upon each attempt to execute the instruction. Each exception service routine should make the appropriate corrections to the instruction and then restart the instruction. In this way, exceptions can be serviced until the instruction executes properly.

The Cyrix III CPU supports instruction restart after all faults, except when an instruction causes a task switch to a task whose task state segment (TSS) is partially not present. A TSS can be partially not present if the TSS is not page aligned and one of the pages where the TSS resides is not currently in memory

.

Table 2-33. Interrupt and Exception Priorities

PRIORITY	DESCRIPTION	NOTES
0	Warm Reset	Caused by the assertion of INIT#.
1	Debug traps and faults from previous instruction.	Includes single-step trap and data breakpoints specified in the debug registers.
2	Debug traps for next instruction.	Includes instruction execution breakpoints specified in the debug registers.
3	Hardware Cache Flush	Caused by the assertion of FLUSH#.
4	SMM hardware interrupt.	SMM interrupts are caused by SMI# asserted and always have highest priority.
5	Non-maskable hardware interrupt.	Caused by NMI asserted.
6	Maskable hardware interrupt.	Caused by INTR asserted and IF = 1.
7	Faults resulting from fetching the next instruction.	Includes segment not present, general protection fault and page fault.
8	Faults resulting from instruction decoding.	Includes illegal opcode, instruction too long, or privilege violation.
9	WAIT instruction and TS = 1 and MP = 1.	Device not available exception generated.
10	ESC instruction and EM = 1 or TS = 1.	Device not available exception generated.
11	Floating point error exception.	Caused by unmasked floating point exception with NE = 1.
12	Segmentation faults (for each memory reference required by the instruction) that prevent transferring the entire memory operand.	Includes segment not present, stack fault, and general protection fault.
13	Page Faults that prevent transferring the entire memory operand.	
14	Alignment check fault.	

2.10.5 Exceptions in Real Mode

Many of the exceptions described in Table 2-33 (Page 2-95) are not applicable in real mode. Exceptions 10, 11, and 14 do not occur in real mode. Other exceptions have slightly different meanings in real mode as listed in Table 2-34.

Table 2-34. Exception Changes in Real Mode

VECTOR NUMBER	PROTECTED MODE FUNCTION	REAL MODE FUNCTION
8	Double fault.	Interrupt table limit overrun.
10	Invalid TSS.	x
11	Segment not present.	x
12	Stack fault.	SS segment limit overrun.
13	General protection fault.	CS, DS, ES, FS, GS segment limit overrun.
14	Page fault.	x

Note: x = does not occur

2.10.6 Error Codes

When operating in protected mode, the following exceptions generate a 16-bit error code:

Double Fault	Invalid TSS
Alignment Check	Segment Not Present
Page Fault	Stack Fault
	General Protection Fault

The error code is pushed onto the stack prior to entering the exception handler. The error code format is shown in Figure 2-34 and the error code bit definitions are listed in Table 2-35. Bits 15-3 (selector index) are not meaningful if the error code was generated as the result of a page fault. The error code is always zero for double faults and alignment check exceptions.

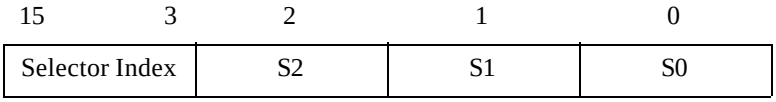


Figure 2-34. Error Code Format

Table 2-35. Error Code Bit Definitions

FAULT TYPE	SELECTOR INDEX (BITS 15-3)	S2 (BIT 2)	S1 (BIT 1)	S0 (BIT 0)
Double Fault or Alignment Check	0	0	0	0
Page Fault	Reserved.	Fault caused by: 0 = not present page 1 = page-level protection violation.	Fault occurred during: 0 = read access 1 = write access.	Fault occurred during: 0 = supervisor access. 1 = user access.
IDT Fault	Index of faulty IDT selector.	Reserved.	1	If = 1, exception occurred while trying to invoke exception or hardware interrupt handler.
Segment Fault	Index of faulty selector.	TI bit of faulty selector.	0	If =1, exception occurred while trying to invoke exception or hardware interrupt handler.

2.11 System Management Mode

System Management Mode (SMM) is a distinct CPU mode that differs from normal CPU x86 operating modes (real mode, V86 mode, and protected mode) and is most often used to perform power management.

The Cyrix III CPU is backward compatible with the SL-compatible SMM found on previous Cyrix microprocessors. On the Cyrix III SMM has been enhanced to optimized software emulation of multimedia and I/O peripherals.

The Cyrix Enhanced SMM provides new features:

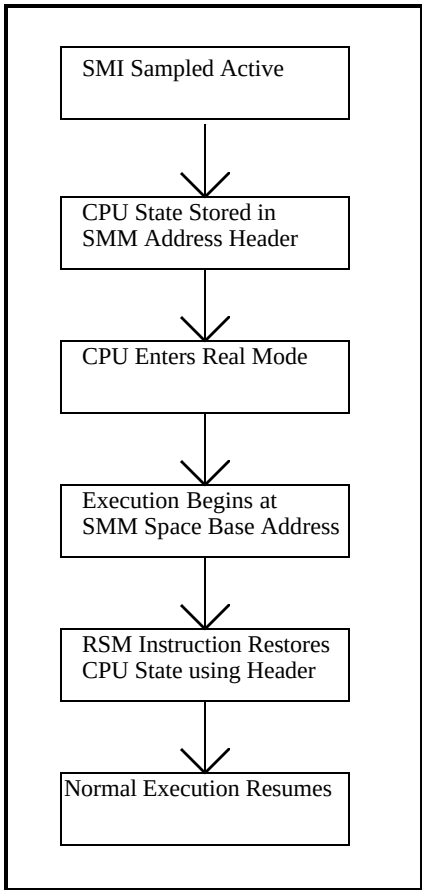
- Cacheability of SMM memory
- Improved SMM entry and exit time.

Overall Operation

The overall operation of a SMM operation is shown in (Figure 2-35). SMM is entered using the System Management Interrupt (SMI) pin. SMI interrupts have higher priority than any other interrupt, including NMI interrupts.

Upon entering SMM mode, portions of the CPU state are automatically saved in the SMM address memory space header. The CPU enters real mode and begins executing the SMI service routine in SMM address space.

Execution of a SMM routine starts at the base address in SMM memory address space. Since the SMM routines reside in SMM memory space, SMM routines can be made totally transparent to all software, including protected-mode operating systems.



SMI Execution Flow Diagram

Figure 2-35. SMI Execution Flow Diagram

2.11.1 SMM Memory Space

SMM memory must reside within the bounds of physical memory and not overlap with system memory. SMM memory space (Figure 2-36) is defined by setting the SM3 bit in CCR1 and specifying the base address and size of the SMM memory space in the ARR3 register.

The base address must be a multiple of the SMM memory space size. For example, a 32 KB SMM memory space must be located on a 32KB address boundary. The memory space size can range from 4 KB to 4 GB SMM accesses ignore the state of the A20M# input pin and drive the A20 address bit to the unmasked value.

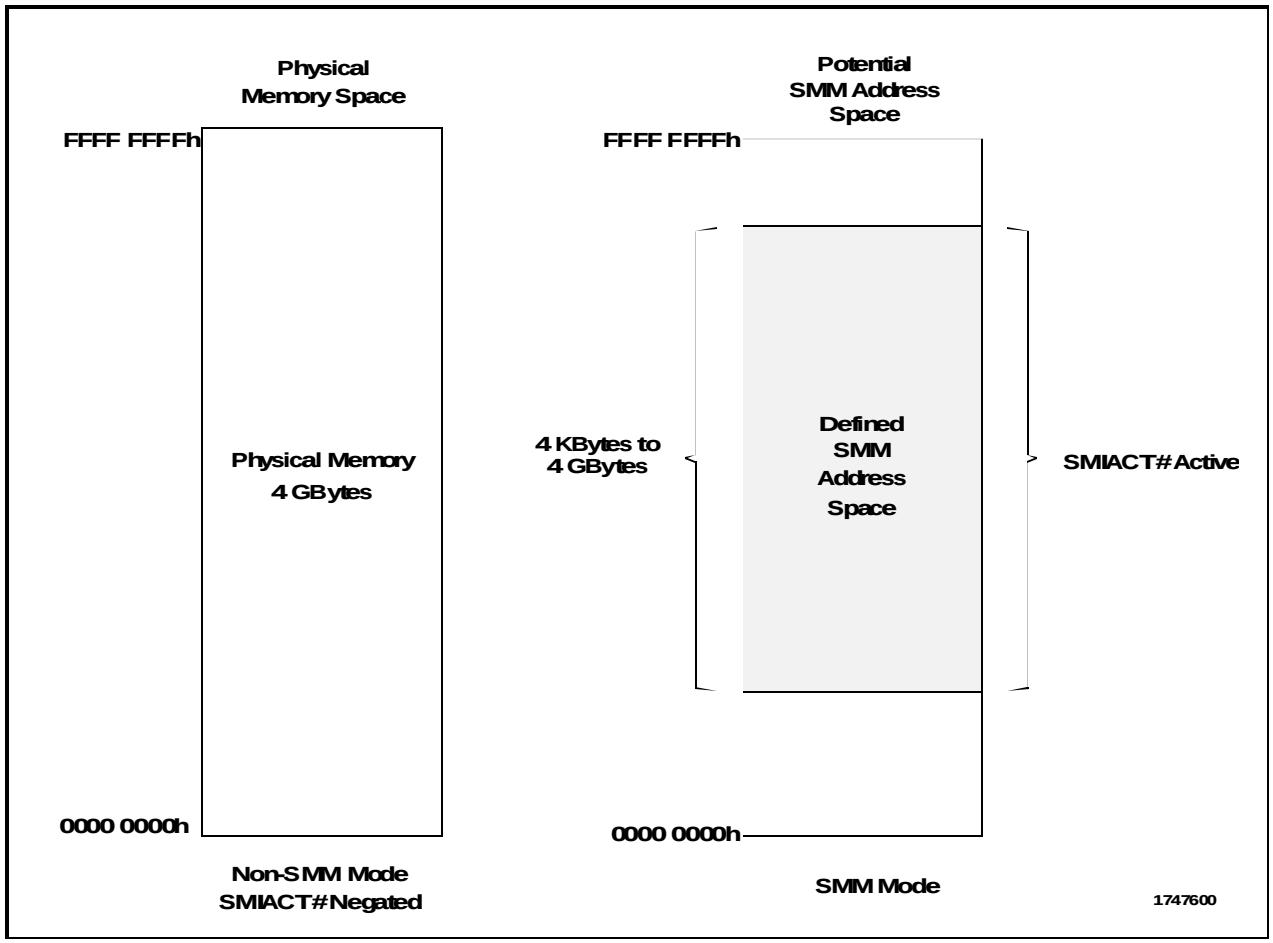


Figure 2-36. System Management Memory Space

2.11.2 SMM Memory SpaceHeader

The SMM Memory Space Header (Figure 2-37) is used to store the CPU state prior to starting an SMM routine. The fields in this header are described in Table 2-36 (Page 2-101). After the SMM routine has completed, the header information is used to restore the original CPU state. The location of the SMM header is determined by the SMM Header Address Register (SMHR).

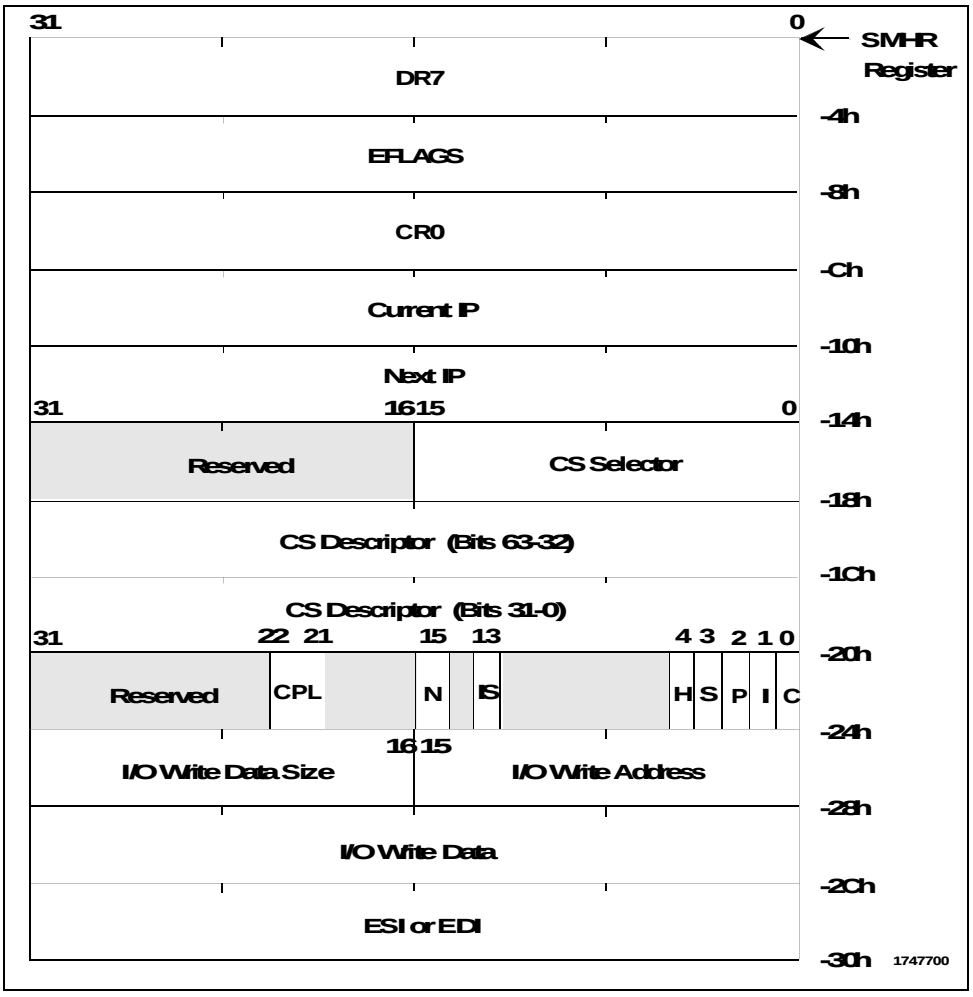


Figure 2-37. SMM Memory Space Header

Table 2-36. SMM Memory Space Header

NAME	DESCRIPTION	SIZE
DR7	The contents of Debug Register 7.	4 Bytes
EFLAGS	The contents of Extended Flags Register.	4 Bytes
CR0	The contents of Control Register 0.	4 Bytes
Current IP	The address of the instruction executed prior to servicing SMI interrupt.	4 Bytes
Next IP	The address of the next instruction that will be executed after exiting SMM mode.	4 Bytes
CS Selector	Code segment register selector for the current code segment.	2 Bytes
CS Descriptor	Code segment register descriptor for the current code segment.	8 Bytes
CPL	Current privilege level for current code segment.	2 Bits
IS	Internal SMI Indicator If IS =1: current SMM is the result of an internal SMI event. If IS =0: current SMM is the result of an external SMI event.	1 Bit
H	SMI during CPU HALT state indicator If H = 1: the processor was in a halt or shutdown prior to servicing the SMM interrupt.	1 Bit
S	Software SMM Entry Indicator. If S = 1: current SMM is the result of an SMINT instruction. If S = 0: current SMM is not the result of an SMINT instruction.	1 Bit
P	REP INSx/OUTSx Indicator If P = 1: current instruction has a REP prefix. If P = 0: current instruction does not have a REP prefix.	1 Bit
I	IN, INSx, OUT, or OUTSx Indicator If I = 1: if current instruction performed is an I/O WRITE. If I = 0: if current instruction performed is an I/O READ.	1 Bit
C	Code Segment writable Indicator If C = 1: the current code segment is writable. If C = 0: the current code segment is not writable.	1 Bit
I/O	Indicates size of data for the trapped I/O write: 01h = byte 03h = word 0Fh = dword	2 Bytes
I/O Write Address	I/O Write Address Processor port used for the trapped I/O write.	2 Bytes
I/O Write Data	I/O Write Data Data associated with the trapped I/O write.	4 Bytes
ESI or EDI	Restored ESI or EDI value. Used when it is necessary to repeat a REP OUTSx or REP INSx instruction when one of the I/O cycles caused an SMI# trap.	4 Bytes

Note: INSx = INS, INSB, INSW or INSD instruction.
Note: OUTSx = OUTS, OUTSB, OUTSW and OUTSD instruction.

Current and Next IP Pointers

Included in the header information are the Current and Next IP pointers. The Current IP points to the instruction executing when the SMI was detected and the Next IP points to the instruction that will be executed after exiting SMM.

Normally after an SMM routine is completed, the instruction flow begins at the Next IP address. However, if an I/O trap has occurred, instruction flow should return to the Current IP to complete the I/O instruction.

If SMM has been entered due to an I/O trap for a REP INSx or REP OUTSx instruction, the Current IP and Next IP fields contain the same address.

If an entry into SMM mode was caused by an I/O trap, the port address, data size and data value associated with that I/O operation are stored in the SMM header. Note that these values are only valid for I/O operations. The I/O data is not restored within the CPU when executing a RSM instruction.

Under these circumstances the I and P bits, as well as ESI/EDI field, contain valid information.

Also saved are the contents of debug register 7 (DR7), the extended flags register (EFLAGS), and control register 0 (CR0).

SMM Header Address Pointer

The SMM Header Address Pointer Register (SMHR) (Figure 2-38) contains the 32-bit SMM Header pointer. The SMHR address is dword aligned, so the two least significant bits are ignored.

The SMHR valid bit (bit 0) is cleared with every write to ARR3 and during a hardware RESET#. Upon entry to SMM, the SMHR valid bit is examined before the CPU state is saved into the SMM memory space header. When the valid bit is reset, the SMM header pointer will be calculated (ARR3 base field + ARR3 size field) and loaded into the SMHR and the valid bit will be set.

If the desired SMM header location is different than the top of SMM memory space, as may be the case when nesting SMI's, then the SMHR register must be loaded with a new value and valid bit from within the SMI routine before nesting is enabled.

The SMM memory space header can be relocated using the new RDSHR and WRSHR instructions.

Figure 2-38. SMHR Register

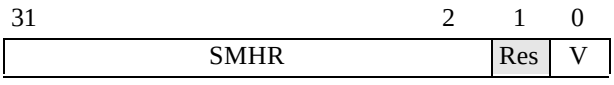


Table 2-37. SMHR Register Bits

BIT POSITION	DESCRPTION
31 - 2	SMHR header pointer address.
1	Reserved
0	Valid Bit

2.11.3 SMM Instructions

After entering the SMI service routine, the MOV, SVDC, SVLDT and SVTS instructions (Table 2-38) can be used to save the complete CPU state information. If the SMI service routine modifies more than what is automatically saved or forces the CPU to power down, the complete CPU state information must be saved. Since the CPU is a static device, its internal state is retained when the input clock is stopped. Therefore, an entire CPU state save is not necessary prior to stopping the input clock.

Table 2-38. SMM Instruction Set

INSTRUCTION	OPCODE	FORMAT	DESCRIPTION
SVDC	0F 78 [mod sreg3 r/m]	SVDC mem80, sreg3	*Save Segment Register and Descriptor* Saves reg (DS, ES, FS, GS, or SS) to mem80.
RSDC	0F 79 [mod sreg3 r/m]	RSDC sreg3, mem80	*Restore Segment Register and Descriptor* Restores reg (DS, ES, FS, GS, or SS) from mem80. Use RSM to restore CS. Note: Processing "RSDC CS, Mem80" will produce an exception.
SVLDT	0F 7A [mod 000 r/m]	SVLDT mem80	*Save LDTR and Descriptor* Saves Local Descriptor Table (LDTR) to mem80.
RSLDT	0F 7B [mod 000 r/m]	RSLDT mem80	*Restore LDTR and Descriptor* Restores Local Descriptor Table (LDTR) from mem80.
SVTS	0F 7C [mod 000 r/m]	SVTS mem80	*Save TSR and Descriptor* Saves Task State Register (TSR) to mem80.
RSTS	0F 7D [mod 000 r/m]	RSTS mem80	*Restore TSR and Descriptor* Restores Task State Register (TSR) from mem80.
SMINT	0F 38	SMINT	*Software SMM Entry* CPU enters SMM mode. CPU state information is saved in SMM memory space header and execution begins at SMM base address.
RSM	0F AA	RSM	*Resume Normal Mode* Exits SMM mode. The CPU state is restored using the SMM memory space header and execution resumes at interrupted point.
RDSHR	0F 36	RDSHR ereg/mem32	*Read SMM Header Pointer Register* Saves SMM header pointer to extended register or memory.
WRSHR	0F 37	WRSHR ereg/mem32	*Write SMM Header Pointer Register* Load SMM header pointer register from extended register or memory.

Note: mem32 = 32-bit memory location
mem80 = 80-bit memory location

The SMM instructions listed in Table 2-38, (except the SMINT instruction) can be executed only if:

1) ARR3 Size > 0
2) Current Privilege Level =0
3) the CPU is executing an SMI service routine.
4) USE_SMI (CCR1- bit 1) = 1
5) SM3 (CCR1-bit 7) = 1

If the above conditions are not met and an attempt is made to execute an SVDC, RSDC, SVLDT, RSLDT, SVTS, RSTS, SMINT, RSM, RDSHR, or WDSHR instruction, an invalid opcode exception is generated. These instructions can be executed outside of defined SMM space provided the above conditions are met.

The SVDC, RSDC, SVLDT, RSLDT, SVTS and RSTS instructions save or restore 80 bits of data, allowing the saved values to include the hidden portion of the register contents.

The WRSHR instruction loads the contents of either a 32-bit memory operand or a 32-bit register operand into the SMHR pointer register based on the value of the mod r/m instruction byte. Likewise the RDSHR instruction stores the contents of the SMHR pointer register to either a 32 bit memory operand or a 32 bit register operand based on the value of the mod r/m instruction byte.

2.11.4 SMM Operation

This section details the SMM operations.

Entering SMM

Entering SMM requires the assertion of the SMI# pin. SMI interrupts have higher priority than any interrupt including NMI interrupts.

For the SMI# to be recognized, the following configuration register bits must be set as shown in Table 2-39.

Table 2-39. Requirements for Recognizing SMI# and SMINT

REGISTER (Bit)		SMI#	SMINT
SMI	CCR1 (1)	1	1
ARR3	SIZE (3-0)	> 0	> 0
SM3	CCR1 (7)	1	1

Upon entry into SMM, after the SMM header has been saved, the CR0, EFLAGS, and DR7 registers are set to their reset values. The Code Segment (CS) register is loaded with the base, as defined by the ARR3 register, and a limit of 4 GB. The SMI service routine then begins execution at the SMM base address in real mode.

Saving the CPU State

The programmer must save the value of any registers that may be changed by the SMI service routine. For data accesses immediately after entering the SMI service routine, the programmer must use CS as a segment override. I/O port access is possible during the routine but care must be taken to save registers modified by the I/O instructions. Before using a segment register, the register and the register's descriptor cache contents should be saved using the SVDC instruction. While executing in the SMM space, execution flow can transfer to normal memory locations.

Program Execution

Hardware interrupts, (INTRs and NMIs), may be serviced during a SMI service routine. If interrupts are to be serviced while executing in the SMM memory space, the SMM memory space must be within the 0 to 1 MB address range to guarantee proper return to the SMI service routine after handling the interrupt.

INTRs are automatically disabled when entering SMM since the IF flag is set to its reset value. Once in SMM, the INTR can be enabled by setting the IF flag. NMI is also automatically disable when entering SMM. Once in SMM, NMI can be enabled by setting NMI_EN in CCR3. If NMI is not enabled, the CPU latches one NMI event and services the interrupt after NMI has been enabled or after exiting SMM through the RSM instruction.

Within the SMI service routine, protected mode may be entered and exited as required, and real or protected mode device drivers may be called.

Exiting SMM

To exit the SMI service routine, a Resume (RSM) instruction, rather than an IRET, is executed. The RSM instruction causes the Cyrix III processor to restore the CPU state using the SMM header information and resume execution at the interrupted point. If the full CPU state was saved by the programmer, the stored values should be reloaded prior to executing the RSM instruction using the MOV, RSDC, RSLDT and RSTS instructions.

When the RSM instruction is executed at the end of the SMI handler, the EIP instruction pointer is automatically read from the NEXT IP field in the SMM header.

When restarting I/O instructions, the value of NEXTIP may need modification. Before executing the RSM instruction, use a MOV instruction to move the CURRENTIP value to the NEXT IP location as the CURRENT IP value is valid if an I/O instruction was executing when the SMI interrupt occurred. Execution is then returned to the I/O instruction, rather than to the instruction after the I/O instruction.

A set H bit in the SMM header indicates that a HLT instruction was being executed when the SMI occurred. To resume execution of the HLT instruction, the NEXTIP field in the SMM header should be decremented by one before executing RSM instruction.

2.11.5 SL and Cyrix SMM Operating Modes

There are two SMM modes, SL-compatible mode (default) and Cyrix SMM mode.

2.11.5.1 SL-Compatible SMM Mode

While in SL-compatible mode, SMM memory space accesses can only occur during an SMI service routine. This includes the time when the SMI service routine accesses memory outside the defined SMM memory space.

SMM memory caching is not supported in SL-compatible SMM mode. If a cache inquiry cycle occurs while in SMM mode, any resulting write-back cycle is issued with SMM_MEM asserted. This occurs even though the write-back cycle is intended for normal memory rather than SMM memory. To avoid this problem it is recommended that the internal caches be flushed prior to servicing an SMI event. Of course in write-back mode this could add an indeterminate delay to servicing of SMI.

An interrupt on the SMI# input pin has higher priority than the NMI input. The SMI# input pin is falling edge sensitive and is sampled on every rising edge of the processor input clock.

Asserting SMI# forces the processor to save the CPU state to memory defined by SMHR register and to begin execution of the SMI service routine at the beginning of the defined SMM memory space. After the processor internally acknowledges the SMI# interrupt, an SMM acknowledge cycle is driven onto the bus, and SMM_MEM (pin EX4) is asserted.

When the RSM instruction is executed, the CPU negates the SMM_MEM 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.

During RESET#, the USE_SMI bit in CCR1 is cleared.

2.11.5.2 Cyrix Enhanced SMM Mode

The Cyrix SMM Mode is enabled when bit0 in the CCR6 (SMM_MODE) is set. Only in Cyrix enhanced SMM mode can SMM space be cached.

Cacheability of SMM Space

In SL-compatible SMM mode, caching is not available, but in Cyrix SMM mode, both code and data caching is supported. In order to cache SMM data and avoid coherency issues the processor assumes no overlap of main memory with SMM memory. This implies that a section of main memory must be dedicated for SMM.

The on-chip cache sets a special ID bit in the cache tag block for each line that contains SMM code data. This ID bit is then used by the bus controller to regulate assertion of the SMM_MEM pin for write-back of any SMM data.

2.11.6 Maintaining the FPU and MMX States

If power will be removed from the CPU or if the SMM routine will execute MMX or FPU instructions, then the MMX or FPU state should

be maintained for the application running before SMM was entered. If the MMX or 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. 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.

2.12 Sleep and Halt

The Halt Instruction (HLT) stops program execution and prevents the processor from using the local bus until restarted. The Cyrix III CPU then issues a special HALT bus cycle and enters a low-power suspend mode if the SUSP_HLT bit in CCR2 is set. SMI, NMI, INTR with interrupts enabled (IF bit in EFLAGS=1), INIT# or RESET# force the CPU out of the halt state. If interrupted, the saved code segment and instruction pointer specify the instruction following the HLT.

Sleep states can be entered using STPCLK# and SLP# pins to perform power management. The three low power states are Stop Grant, Sleep, and Deep Sleep.

Stop Grant state is entered by asserting STPCLK#. When STPCLK# is deasserted, the Stop Grant state is exited and processor returns to normal operation.

Sleep state is entered by asserting the SLP# pin while the processor is in Stop Grant state. Sleep state is exited when SLP# is deasserted and the processor returns to Stop Grant state.

Deep Sleep can be entered while in Sleep state by stopping the bus clock pin BCLK. When BCLK is restarted the processor exits Deep Sleep and returns to Sleep state.

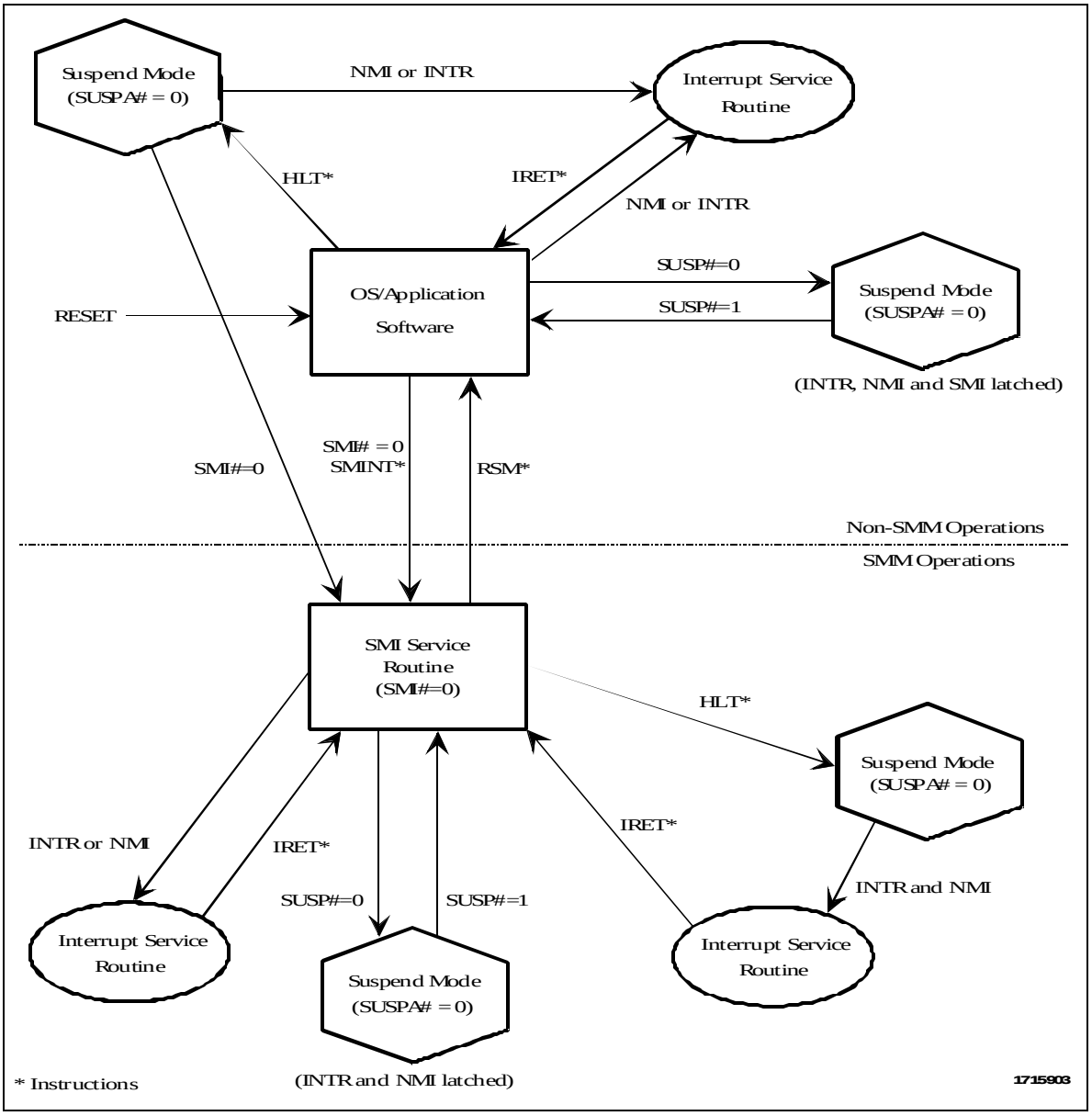


Figure 2-39. SMM and Suspend Mode State Diagram

2.13 Protection

Segment protection and page protection are safeguards built into the Cyrix III CPU protected mode architecture which deny unauthorized or incorrect access to selected memory addresses. These safeguards allow multitasking programs to be isolated from each other and from the operating system. Page protection is discussed earlier in this chapter. This section concentrates on segment protection.

Selectors and descriptors are the key elements in the segment protection mechanism. The segment base address, size, and privilege level are established by a segment descriptor. Privilege levels control the use of privileged instructions, I/O instructions and access to segments and segment descriptors. Selectors are used to locate segment descriptors.

Segment accesses are divided into two basic types, those involving code segments (e.g., control transfers) and those involving data accesses. The ability of a task to access a segment depends on the:

- Segment type
- Instruction requesting access
- Type of descriptor used to define the segment
- Associated privilege levels (described below).

Data stored in a segment can be accessed only by code executing at the same or a more privileged level. A code segment or procedure can only be called by a task executing at the same or a less privileged level.

2.13.1 Privilege Levels

The values for privilege levels range between 0 and 3. Level 0 is the highest privilege level (most privileged), and level 3 is the lowest privilege level (least privileged). The privilege level in real mode is effectively 0.

The Descriptor Privilege Level (DPL) is the privilege level defined for a segment in the segment descriptor. The DPL field specifies the minimum privilege level needed to access the memory segment pointed to by the descriptor.

The Current Privilege Level (CPL) is defined as the current task's privilege level. The CPL of an executing task is stored in the hidden portion of the code segment register and essentially is the DPL for the current code segment.

The Requested Privilege Level (RPL) specifies a selector's privilege level and is used to distinguish between the privilege level of a routine actually accessing memory (the CPL), and the privilege level of the original requestor (the RPL) of the memory access. The lesser of the RPL and CPL is called the effective privilege level (EPL). Therefore, if RPL = 0 in a segment selector, the effective privilege level is always determined by the CPL. If RPL = 3, the effective privilege level is always 3 regardless of the CPL.

For a memory access to succeed, the effective privilege level (EPL) must be at least as privileged as the descriptor privilege level (EPL ≤ DPL). If the EPL is less privileged than the DPL (EPL > DPL), a general protection fault is generated. For example, if a segment has a DPL = 2, an instruction accessing the segment only succeeds if executed with an EPL ≤ 2.

2.13.2 I/O Privilege Levels

The I/O Privilege Level (IOPL) allows the operating system executing at CPL=0 to define the least privileged level at which IOPL-sensitive instructions can unconditionally be used. The IOPL-sensitive instructions include CLI, IN, OUT, INS, OUTS, REP INS, REP OUTS, and STI. Modification of the IF bit in the EFLAGS register is also sensitive to the I/O privilege level. The IOPL is stored in the EFLAGS register.

An I/O permission bit map is available as defined by the 32-bit Task State Segment (TSS). Since each task can have its own TSS, access to individual processor I/O ports can be granted through separate I/O permission bit maps.

If CPL ≤ IOPL, IOPL-sensitive operations can be performed. If CPL > IOPL, a general protection fault is generated if the current task is associated with a 16-bit TSS. If the current task is associated with a 32-bit TSS and CPL > IOPL, the CPU consults the I/O permission bitmap in the TSS to determine on a port-by-port basis whether or not I/O instructions (IN, OUT, INS, OUTS, REP INS, REP OUTS) are permitted, and the remaining IOPL-sensitive operations generate a general protection fault.

2.13.3 Privilege Level Transfers

A task's CPL can be changed only through intersegment control transfers using gates or task switches to a code segment with a different privilege level. Control transfers result from exception and interrupt servicing and from execution of the CALL, JMP, INT, IRET and RET instructions.

There are five types of control transfers that are summarized in Table 2-40 (Page 2-111). Control transfers can be made only when the operation causing the control transfer references the correct descriptor type. Any violation of these descriptor usage rules causes a general protection fault.

Any control transfer that changes the CPL within a task results in a change of stack. The initial values for the stack segment (SS) and stack pointer (ESP) for privilege levels 0, 1, and 2 are stored in the TSS. During a CALL control transfer, the SS and ESP are loaded with the new stack pointer and the previous stack pointer is saved on the new stack. When returning to the original privilege level, the RET or IRET instruction restores the less-privileged stack

Table 2-40. Descriptor Types Used for Control Transfer

TYPE OF CONTROL TRANSFER	OPERATION TYPES	DESCRIPTOR REFERENCED	DESCRIPTOR TABLE
Intersegment within the same privilege level.	JMP, CALL, RET, IRET*	Code Segment	GDT or LDT
Intersegment to the same or a more privileged level. Interrupt within task (could change CPL level).	CALL	Gate Call	GDT or LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a less privileged level (changes task CPL).	RET, IRET*	Code Segment	GDT or LDT
Task Switch via TSS	CALL, JMP	Task State Segment	GDT
Task Switch via Task Gate	CALL, JMP	Task Gate	GDT or LDT
	IRET**, Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

* NT (Nested Task bit in EFLAGS) = 0
** NT (Nested Task bit in EFLAGS) = 1

Gates

Gate descriptors provide protection for privilege transfers among executable segments. Gates are used to transition to routines of the same or a more privileged level. Call gates, interrupt gates and trap gates are used for privilege transfers within a task. Task gates are used to transfer between tasks.

Gates conform to the standard rules of privilege. In other words, gates can be accessed by a task if the effective privilege level (EPL) is the same or more privileged than the gate descriptor's privilege level (DPL).

2.13.4 Initialization and Transition to Protected Mode

The Cyrix III processor switches to real mode immediately after RESET#. While operating in real mode, the system tables and registers should be initialized. The GDTR and IDTR must point to a valid GDT and IDT, respectively. The GDT must contain descriptors which describe the initial code and data segments.

The processor can be placed in protected mode by setting the PE bit in the CR0 register. After enabling protected mode, the CS register should be loaded and the instruction decode queue should be flushed by executing an intersegment JMP. Finally, all data segment registers should be initialized with appropriate selector values.

2.14 Virtual 8086 Mode

Both real mode and virtual 8086 (V86) mode are supported by the Cyrix III CPU allowing execution of 8086 application programs and 8086 operating systems. V86 mode allows the execution of 8086-type applications, yet still permits use of the Cyrix III CPU paging mechanism. V86 tasks run at privilege level 3. When loaded, all segment limits are set to FFFFh (64K) as in real mode.

2.14.1 V86 Memory Addressing

While in V86 mode, segment registers are used in an identical fashion to real mode. The contents of the segment register are multiplied by 16 and added to the offset to form the segment base linear address. The Cyrix III CPU permits the operating system to select which programs use the V86 address mechanism and which programs use protected mode addressing for each task.

The Cyrix III CPU also permits the use of paging when operating in V86 mode. Using paging, the 1-MB memory space of the V86 task can be mapped to anywhere in the 4-GB linear memory space of the Cyrix III CPU.

The paging hardware allows multiple V86 tasks to run concurrently, and provides protection and operating system isolation. The paging hardware must be enabled to run multiple V86tasks or to relocate the address space of a V86task to physical address space greater than 1MB.

2.14.2 V86 Protection

All V86 tasks operate with the least amount of privilege (level 3) and are subject to all of the Cyrix III CPU protected mode protection checks. As a result, any attempt to execute a privileged instruction within a V86 task results in a general protection fault.

In V86 mode, a slightly different set of instructions are sensitive to the I/O privilege level (IOPL) than in protected mode. These instructions are: CLI, INT n, IRET, POPF, PUSHF, and STI. The INT3, INTO and BOUND variations of the INT instruction are not IOPL sensitive.

2.14.3 V86 Interrupt Handling

To fully support the emulation of an 8086-type machine, interrupts in V86 mode are handled as follows. When an interrupt or exception is serviced in V86 mode, program execution transfers to the interrupt service routine at privilege level 0 (i.e., transition from V86 to protected mode occurs) and the VM bit in the EFLAGS register is cleared. The protected mode interrupt service routine then determines if the interrupt came from a protected mode or V86 application by examining the VM bit in the EFLAGS image stored on the stack. The interrupt service routine may then choose to allow the 8086 operating system to handle the interrupt or may emulate the function of the interrupt handler. Following completion of the interrupt service routine, an IRET instruction restores the EFLAGS register (restores VM=1) and segment selectors and control returns to the interrupted V86 task.

2.14.4 Entering and Leaving V86 Mode

V86 mode is entered from protected mode by either executing an IRET instruction at CPL = 0 or by task switching. If an IRET is used, the stack must contain an EFLAGS image with VM = 1. If a task switch is used, the TSS must contain an EFLAGS image containing a 1 in the VM bit position. The POPF instruction cannot be used to enter V86 mode since the state of the VM bit is not affected. V86 mode can only be exited as the result of an interrupt or exception. The transition out must use a 32-bit trap or interrupt gate which must point to a non-conforming privilege level 0 segment (DPL = 0), or a 32-bit TSS. These restrictions are required to permit the trap handler to IRET back to the V86 program.

2.15 Floating Point Unit Operations

The Cyrix III CPU includes an on-chip FPU that provides the user access to a complete set of floating point instructions (see Chapter 6). Information is passed to and from the FPU using eight data registers accessed in a stack-like manner, a control register, and a status register. The Cyrix III CPU also provides a data register tag word which improves context switching and performance by maintaining empty/non-empty status for each of the eight data registers. In addition, registers in the CPU contain pointers to (a) the memory location containing the current instruction word and (b) the memory location containing the operand associated with the current instruction word (if any).

FPU Tag Word Register. The Cyrix III CPU maintains a tag word register (Figure 2-40 (Page 2-114)) comprised of two bits for each physical data register. Tag Word fields assume one of four values depending on the contents of their associated data registers, Valid (00), Zero (01), Special (10), and Empty (11). Note: Denormal, Infinity, QNaN, SNaN and unsupported formats are tagged as “Special”. Tag values are maintained transparently by the Cyrix III CPU and are only available to the programmer indirectly through the FSTENV and FSAVE instructions.

FPU Control and Status Registers. The FPU circuitry communicates information about its status and the results of operations to the programmer via the status register. The FPU status register is comprised of bit fields that reflect exception status, operation execution status, register status, operand class, and comparison results. The FPU status register bit

definitions are shown in Figure 2-41 (Page 2-114) and Table 2-41 (Page 2-114).

The FPU Mode Control Register (MCR) is used by the CPU to specify the operating mode of the FPU. The MCR contains bit fields which specify the rounding mode to be used, the precision by which to calculate results, and the exception conditions which should be reported to the CPU via traps. The user controls precision, rounding, and exception reporting by setting or clearing appropriate bits in the MCR. The FPU mode control register bit definitions are shown in Figure 2-42 (Page 2-115) and Table 2-42 (Page 2-115).

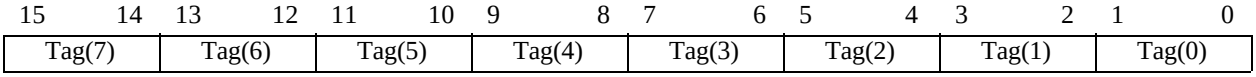


Figure 2-40. FPU Tag Word Register

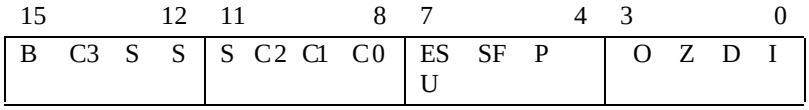


Figure 2-41. FPU Status Register

Table 2-41. FPU Status Register Bit Definitions

BIT POSITION	NAME	DESCRIPTION
15	B	Copy of the ES bit. (ES is bit 7 in this table.)
14, 10 - 8	C3 - C0	Condition code bits.
13 - 11	SSS	Top of stack register number which points to the current TOS.
7	ES	Error indicator. Set to 1 if an unmasked exception is detected.
6	SF	Stack Fault or invalid register operation bit.
5	P	Precision error exception bit.
4	U	Underflow error exception bit.
3	O	Overflow error exception bit.
2	Z	Divide by zero exception bit.
1	D	Denormalized operand error exception bit.
0	I	Invalid operation exception bit.

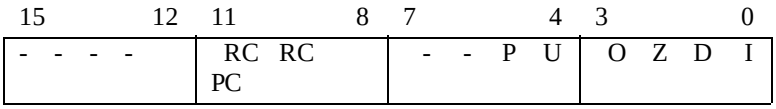


Figure 2-42. FPU Mode Control Register

Table 2-42. FPU Mode Control Register Bit Definitions

BIT POSITION	NAME	DESCRIPTION
11 - 10	RC	Rounding Control bits: 00 Round to nearest or even 01 Round towards minus infinity 10 Round towards plus infinity 11 Truncate
9 - 8	PC	Precision Control bits: 00 24-bit mantissa 01 Reserved 10 53-bit mantissa 11 64-bit mantissa
5	P	Precision error exception bit mask.
4	U	Underflow error exception bit mask.
3	O	Overflow error exception bit mask.
2	Z	Divide by zero exception bit mask.
1	D	Denormalized operand error exception bit mask.
0	I	Invalid operation exception bit mask.

2.16 MMX Operations

The Cyrix III CPU provides user access to the MMX instruction set. MMX data is configured in one of four MMX data formats. During operations eight 64-bit MMX registers are utilized.

2.16.1 MMX Data Formats

The MMX instructions operate on 64-bit data groups called “packed data.” A single packed data group can be interpreted as a:

- Packed byte (8 bytes)
- Packed word (4 words)
- Packed doubleword (2 doublewords)
- Quadword (1 quadword)

The packed data types supported are signed and unsigned integer.

2.16.2 MMX Registers

The MMX instruction set operates on eight 64-bit, general-purpose registers (MM0-MM7). These registers are overlaid with the floating point register stack, so no new architectural state is defined by the MMX instruction set. Existing mechanisms for saving and restoring floating point state automatically work for saving and restoring MMX state.

2.16.3 MMX Instruction Set

The MMX instructions operate on all the elements of a signed or unsigned packed data group. All data elements (bytes, words, doublewords or a quadword) are operated on separately in parallel. For example, eight bytes in one packed data group can be added to another packed data group, such that eight independent byte additions are performed in parallel.

2.16.4 Instruction Group Overview

The 57 MMX instructions are grouped into seven categories:

- Arithmetic Instructions
- Comparison Instructions
- Conversion Instructions
- Logical Instructions
- Shift Instructions
- Data Transfer Instructions
- Empty MMX State (EMMS) Instruction

2.16.5 Saturation Arithmetic

For saturating MMX instructions, a ceiling is placed on an overflow and a floor is placed on an underflow. When the result of an operation exceeds the range of the data-type it saturates to the maximum value of the range. Conversely, when a result that is less than the range of a data type, the result saturates to the minimum value of the range.

The saturation limits are shown in Table 2-43.

Table 2-43. Saturation Limits

DATA TYPE	LOWER LIMIT		UPPER LIMIT	
Signed Byte	80h	-128	7Fh	127
Signed Word	8000h	-32,768	7FFFh	32,767

Table 2-43. Saturation Limits

DATA TYPE	LOWER LIMIT		UPPER LIMIT	
Unsigned Byte	00h	0	FFh	256
Unsigned Word	0000h	0	FFFFh	65,535

MMX instructions do not indicate overflow or underflow occurrence by generating exceptions or setting flags.

2.16.6 EMMS Instruction

The EMMS Instruction clears the TOS pointer and sets the entire FPU tag word as empty. An EMMS instruction should be executed at the end of each MMX routine.

3 Cyrix III BUS INTERFACE

The signals used in the Cyrix III CPU bus interface are described in this chapter. Figure 3-1 shows the signal directions and groups the signals for later description. Individual signal are described in Table 3-1 (Page 3-120).

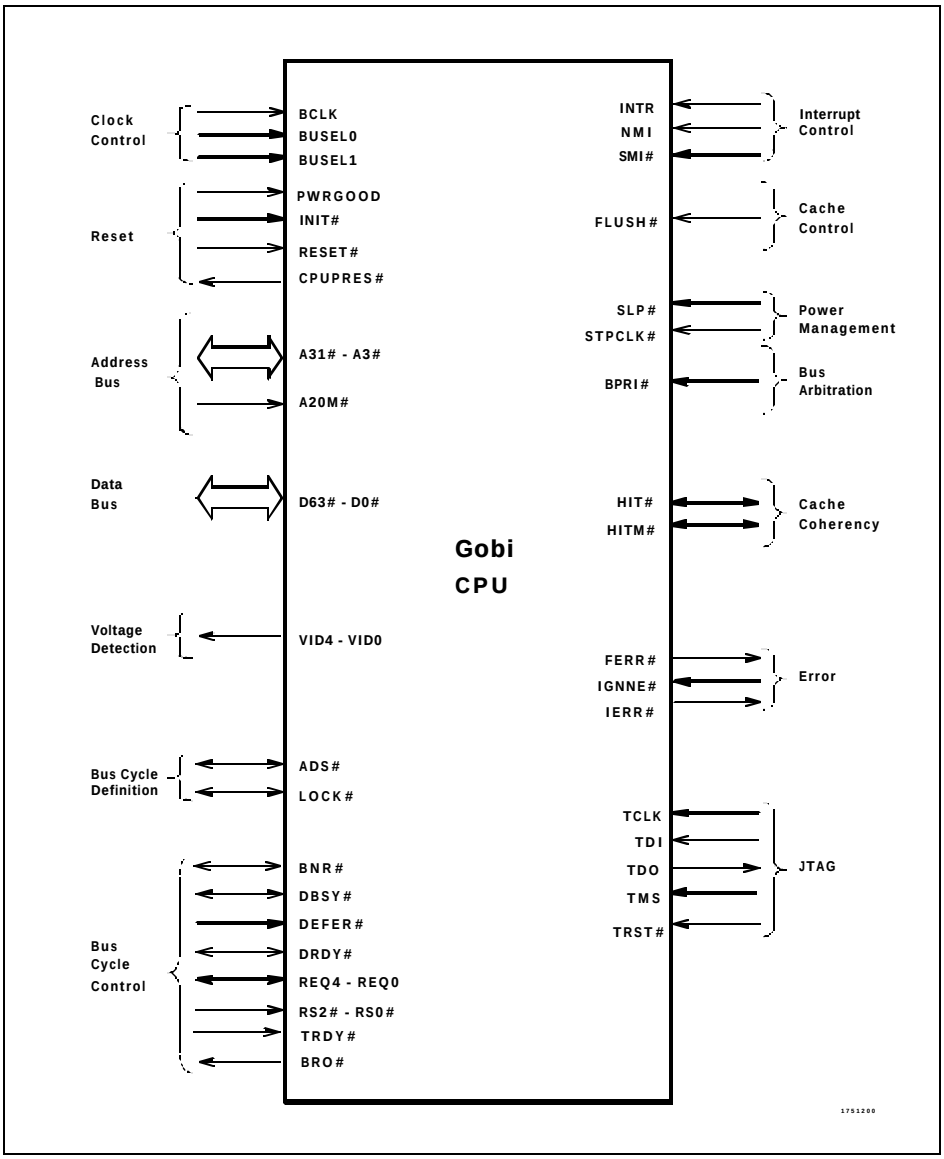


Figure 3-1. Cyrix III CPU Functional Signal Groupings

3.1 Signal Description Table

The Signal Summary Table (Table 3-1) describes the signals in their active state unless otherwise mentioned. Signals ending with a "#" character are active low.

Table 3-1. Cyrix III CPU Signals Sorted by Signal Name

Pin Name	Description	I/O	Clock
A[31 -3]#	The Address Bus provides addresses for physical memory and external I/O devices. During cache inquiry cycles, A31#-A3# are used as inputs to perform snoop cycles.	I/O (GTL+)	BCLK
A20M#	A20 Mask causes the CPU to mask (force to 0) the A20 address bit when driving the external address bus or performing an internal cache access. A20M# is provided to emulate the 1 MByte address wrap-around that occurs on the 8086. Snoop addressing is not affected.	I (2.5V)	Asynch
ADS#	Address Strobe begins a memory/I/O cycle and indicates the address bus (A31#-A3#) and transaction request signals (REQ#) are valid.	I/O (GTL+)	BCLK
BCLK	Bus Clock provides the fundamental timing for the Cyrix III CPU. The frequency of the Cyrix III CPU input clock determines the operating frequency of the CPU's bus. External timing is defined referenced to the rising edge of CLK.	I (2.5V)	--
BNR#	Block Next Request signals a bus stall by a bus agent unable to accept new transactions.	I/O (GTL+)	BCLK
BPRI#	Priority Agent Bus Request arbitrates for ownership of the system bus.	I (GTL+)	BCLK
BSEL[0 - 1]	Bus Selection Bus provides system bus frequency data to the CPU.	I (GTL+)	BCLK
BR0#	**Bus Request** Always asserted since Cyrix III supports only uni-processing.	Ground	None
CPUPRES#	CPU Present provides a ground to allow the motherboard to detect the cpu	Ground	None
D[63 - 0]#	Data Bus signals are bi-directional signals which provide the data path between the Cyrix III CPU and external memory and I/O devices. The data bus driver must assert DRDY# to indicate valid data transfer.	I/O (GTL+)	BCLK
DBSY#	Data Bus Busy is asserted by the data bus driver to indicate data bus is in use.	I/O (GTL+)	BCLK
DEFER#	Defer is asserted by target agent (e.g., north bridge) and indicates the transaction cannot be guaranteed as an in-order completion.	I (GTL+)	BCLK
DRDY#	Data Ready is asserted by data driver to indicate that a valid signal is on the data bus.	I/O (GTL+)	BCLK
FERR#	FPU Error Status indicates an unmasked floating point error has occurred. FERR# is asserted during execution of the FPU instruction that caused the error.	O (2.5V)	Asynch

Table 3-1. Cyrix III CPU Signals Sorted by Signal Name

Pin Name	Description	I/O	Clock
FLUSH#	Flush Internal Caches writing back all data in the modified state.	I (2.5V)	Asynch
HIT#	Snoop Hit indicates that the current cache inquiry address has been found in the cache (exclusive or shared states).	I/O (GTL+)	BCLK
HITM#	Snoop Hit Modified indicates that the current cache inquiry address has been found in the cache and dirty data exists in the cache line (modified state).	I/O (GTL+)	BCLK
IERR#	**Internal Error** Tied to a pull-up resistor. Never occurs.	Pull-up.	None
IGNNE#	Ignore Numeric Error forces the Cyrix III CPU to ignore any pending unmasked FPU errors and allows continued execution of floating point instructions.	I (2.5V)	Asynch
INIT#	Initialization resets integer registers and does not effect internal cache or floating point registers.	I (2.5V)	Asynch
INTR	Maskable Interrupt	I (2.5V)	Asynch
NMI	Non-Maskable Interrupt	I (2.5V)	Asynch
LOCK#	Lock Status is used by the CPU to signal to the target that the operation is atomic.	I/O (GTL+)	BCLK
PWRGOOD	Power Good indicates to the CPU that clocks and power supplies are stable and in specification.	I (2.5V)	Asynch
REQ4# - REQ0#	Request Command is asserted by bus driver to define current transaction type.	I/O (GTL+)	BCLK
RESET#	Resets the processor and invalidates internal cache without writing back.	I (GTL+)	BCLK
RS[2 - 0]#	Response Status signals the completion status of the current transaction when the CPU is the response agent.	I (GTL+)	BCLK
SLP#	Sleep, when asserted in the stop grant state, causes the CPU to enter the sleep state.	I (2.5V)	Asynch
SMI#	System Management (SMM) Interrupt forces the processor to save the CPU state to the top of SMM memory and to begin execution of the SMI service routine at the beginning of the defined SMM memory space. An SMI is a higher-priority interrupt than an NMI.	I (2.5V)	Asynch
STPCLK#	Stop Clock causes the CPU to enter the stop grant state.	I (2.5V)	Asynch
TCLK	Test Clock (JTAG) is the clock input used by the Cyrix III CPU's boundary scan (JTAG) test logic. (Called TCK by Intel)	I (2.5V)	--
TDI	Test Data In (JTAG) is the serial data input used by the Cyrix III CPU's boundary scan (JTAG) test logic.	I (2.5V)	TCLK
TDO	Test Data Out (JTAG) is the serial data output used by the Cyrix III CPU's boundary scan (JTAG) test logic.	O (2.5V)	TCLK
TMS	Test Mode Select (JTAG) is the control input used by the Cyrix III CPU's boundary scan (JTAG) test logic.	I (2.5V)	TCLK

Signal Description Table

Table 3-1. Cyrix III CPU Signals Sorted by Signal Name

Pin Name	Description	I/O	Clock
TRDY#	Target Ready indicates that the target is ready to receive a write or write-back transfer from the CPU.	I (GTL+)	BCLK
TRST#	Test Mode Reset (JTAG) initializes the Cyrix III CPU's boundary scan (JTAG) test logic.	I (2.5V)	Asynch
VID[4 - 0]	Voltage Identification Bus informs the regulator system on the motherboard of the CPU Core voltage requirements.	O (2.5V)	Asynch

Table 3-2. Intel® Celeron™ Signals Not Supported by Cyrix® Cyrix III CPU

Pin Name	Description	Reason
BP[3:2]#	Breakpoint.	Debug extension not supported.
BPM[1:0]#	Breakpoint and performance monitor.	Debug extension not supported.
BR[0]#	Bus request.	Multiprocessing not supported.
IERR#	Internal error.	Internal error detection not supported.
PICCLK	Advanced Programmable Interrupt Controller (APIC) clock.	Multiprocessor mode and therefore the APIC bus is not required.
PICD[1:0]	Advanced Programmable Interrupt Controller (APIC) data.	
PRDY#	Probe ready.	Intel debug tools are not supported.
PREQ#	Probe request.	
THERMTRIP#	Thermal Sensor.	Over temperature trip signal is not supported, The CPU does have a thermal diode but supports the other temperature related signals.

3.2 Signal Descriptions

The following paragraphs provide additional information about the Cyrix III CPU signals. For ease of this discussion, the signals are divided into 16 functional groups as illustrated in Figure 3-1 (Page 3-119).

3.2.1 Bus Clock

The Bus Clock Input (BCLK) signal, supplied by the system, is the timing reference used by the Cyrix III CPU bus interface. All external timing parameters are defined with respect to the BCLK rising edge. The BCLK signal enters the Cyrix III CPU where it is multiplied to produce the Cyrix III CPU internal clock signal. During power on, the CLK signal must be running even if CLK does not meet AC specifications.

3.2.2 Acquiring the CPU/Bus Clock Ratio

The CPU/Bus clock ratio is acquired from the group of input signals listed in Table 3-3 (Page 3-125) during reset. Unlike the Intel Celeron processor which hardwires CPU core clock ratios, the Cyrix III processor uses this original Pentium II method for setting the core clock ratio.

The Cyrix III CPU samples NMI, INTR, A20M# and IGNNE# while RESET# is asserted (low) and latches the values at the rising edge of RESET#. These signals must be stable for 1ms prior to the rising edge of RESET# to ensure proper operation.

There is a pin called external ratio pin, EXTRATIO#, which must be low for the clock ratio jumpers to work. This pin defaults to high in the cpu, so if not hooked up, or hooked up to Vcc, the clock ratio jumpers will not work and the cpu default will boot up.

Clock ratios under 2.5:1 and selections identified as not applicable (N/A) are not supported. All clock ratios and frequencies listed may not be available.

A BIOS method for setting the clock ratio is also available. See chapter 2 for the configuration register programming method of setting the clock multiplier. When using the BIOS method only, no jumpers, then EXTRATIO# must be tied high, so that invalid information on the jumper pins is not latched into the cpu before BIOS programming can take place.

Thus there are three ways to set the bus clock multiplier: With board jumpers, with BIOS, or simply using cpu default.

Table 3-3. CPU/Bus Frequency Ratio Encoding

CPU/BUS Ratio	NMI	INTR	A20M#	IGNNE#	Core Block (MHz)		
					66 MHz Bus	100 MHz Bus	133 MHz Bus
2.5	L	H	L	L			333
3.0	L	L	L	H		300	400
3.5	L	H	L	H		350	466
4.0	L	L	H	L		400	
4.5	L	H	H	L	300	450	
5.0	L	L	H	H	333	500	
5.5	L	H	H	H	366		
6.0	H	L	L	L	400		
6.5	H	H	L	L	433		
7.0	H	L	L	H	466		
7.5	H	H	L	H	500		

3.2.3 Providing CPU Voltage Information

The CPU supply voltage requirements are provided by the Voltage Identification Bus (VID) Bus. This bus consists of five CPU pins VID[4 - 0]. The encoding of these pins is listed in Table 3-4.

Note that an “L” in the table indicates the output pin is switched to ground, and a “H” indicates that the CPU pin is open. The VID bus identifies the same voltages on the Celeron from 1.80 to 2.05 volts inclusive. Other encodings are reserved.

Table 3-4. Voltage Identification Bus

Selected CPU Core Voltage in Volts	VID4	VID3	VID2	VID1	VID0
1.80	L	L	H	L	H
1.85	L	L	H	L	L
1.90	L	L	L	H	H
1.95	L	L	L	H	L
2.00	L	L	L	L	H
2.05	L	L	L	L	L
2.10	H	H	H	H	L
2.20	H	H	H	L	H

3.2.4 Acquiring Bus Speed Information

The CPU acquires the bus by sampling pins BSEL0 and BSEL1 pins during reset. The bus speed information is used to setup the internal clocks within the CPU and tune the I/O bus interface. The Cyrix III processor currently supports 66 MHz, 100 MHz and 133 MHz bus frequencies.

Table 3-5. Bus Speed Signaling

BSEL1	BSEL0	Bus Frequency (MHz)
X	L	66
L	H	100
H	H	133

3.2.5 Reset Control

The Cyrix III CPU output signals are initialized to their reset states during the CPU reset sequence.

Asserting RESET# suspends all operations in progress and places the Cyrix III CPU in a reset state. RESET# can be asserted asynchronously but must be de-asserted synchronously from the clock.

On system power-up, RESET# must be held asserted for at least 1 millisecond after PWRGOOD, Vcc and CLK have reached specified DC and AC limits. This delay allows the CPU's clock circuit to stabilize and guarantees proper completion of the reset sequence.

During normal operation, RESET# must be asserted for at least 1 microsecond in order to guarantee the proper reset sequence is executed.

3.2.6 Address Bus

The Address Bus (A[31-3]#) lines provide the physical memory and external I/O device addresses. A[31-3]# are bi-directional signals used by the Cyrix III CPU to drive addresses to both memory devices and I/O devices. During cache inquiry cycles the Cyrix III CPU receives addresses from the system using signals A[31-3]#.

Using signals A[31-3]#, the Cyrix III CPU can address a 4-GByte memory address space. Using signals A[15-3]#, the Cyrix III CPU can address a 64-KByte I/O space through the processor's I/O ports. During I/O accesses, signals A[31-16]# are driven low.

Address Bit 20 Mask (A20M#) is an active low input which causes the Cyrix III CPU to mask (force low) physical address bit 20 when driving the external address bus or when performing an internal cache access. Asserting A20M# emulates the 1 MByte address wrap-around that occurs on the 8086. The A20 signal is never masked during write-back cycles, inquiry cycles, system management address space accesses or when paging is enabled, regardless of the state of the A20M# input.

3.2.7 Data Bus

Data Bus (D63#-D0#) lines carry, bi-directional signals between the Cyrix III CPU and the system (i.e., external memory and I/O devices). The data bus transfers data to the Cyrix III CPU during memory read, I/O read, and interrupt acknowledge cycles. Data is transferred from the Cyrix III CPU during memory and I/O write cycles.

Data setup and hold times must be met for correct read cycle operation. The data bus is driven only while a write cycle is active.

3.2.8 Bus Cycle Definition

A bus transaction is divided into six phases: arbitration phase, request phase, error phase, snoop phase, response phase, and data phase. Each phases is assigned a different set of processor pins.

3.2.8.1 Arbitration Phase

Three CPU pins are used during the arbitration phase:

- Priority Agent Bus Request (BPRI#)
- Block Next Request (BNR#)
- Lock (LOCK#).

When the chipset requires the bus, a request for bus ownership is generated by BPRI# on the chipset. After BPRI# is asserted, the chipset is assigned as the priority agent and will be granted ownership of the next available transaction.

If the P6 bus agent (e.g., CPU or north bridge) anticipates that the system resources, such as the address and data buffers, are going to be temporarily busy, BNR# can be asserted to delay the next transaction. During the bus stall no agent will be granted bus control.

During non-interruptible sequences of bus transactions, LOCK# is asserted to prevent other agents from accessing the bus. This process allows the processor to maintain control of the bus for a series of transactions.

3.2.8.2 Request Phase

During the Request phase the following signals are used:

- Address Strobe (ADS#)
- Request Bus (REQ[4-0]#)
- Address Bus (A[31-3]#)

The Cyrix III asserts ADS# when it is ready to begin the transaction. REQ[4-0]# and A[31-3]# are valid in the clock cycle and one clock after in which ADS# is asserted.

The REQ[4-0]# pins define the type of transaction according to Table 3-6.

Table 3-6. Transaction Types

REQ4#	REQ3#	REQ2#	REQ1#	REQ0#	Transaction Type
0	0	0	0	0	Deferred Reply
0	0	0	0	1	Reserved
0	1	0	0	0	Interrupt Acknowledge or Special Cycle
0	1	0	0	1	Reserved
1	0	0	0	0	I/O Read
1	0	0	0	1	I/O Write
1	1	0	0	x	Reserved
x	x	0	1	0	Memory Read and Invalidate
x	x	0	1	1	Reserved
x	x	1	0	0	Memory Code Read
x	x	1	1	0	Memory Data Read
x	x	1	0	1	Memory Write (no retry)
x	x	1	1	1	Memory Write (may retry)

3.2.8.3 Error Phase

The CPU is passive during Error Phase.

3.2.8.4 Snoop Phase

The snoop signals are:

- Cache Hit (HIT#)
- Cache Hit Modified (HITM#),
- Transaction Defer (DEFER).

HIT# and HITM# are used to maintain cache coherency. The snooped agent asserts HIT# when a snooped line is in the exclusive or shared state in the cache. HITM# is asserted when the line is in the modified state in the cache.

If HIT# and HITM# are asserted together, it indicates that the caching agent is not ready to indicate the snoop status and the snoop phase must be extended.

DEFER# indicates that the transaction cannot be guaranteed to complete in order. The response agent will either be issuing a retry or defer transaction in the response phase.

3.2.8.5 Response Phase

The response pins are:

- Response Status (RS2-0)
- Target Ready (TRDY#).

RS[2-0] provides the response status from the response agent. The status of the response to the transaction request according to RS is listed in the following table:

Table 3-7. Response Type

RS2#	RS1#	RS0#	Response Type
0	0	0	Idle. This is the default state when no response is being delivered.
0	0	1	Retry. Request agent must try the transaction later.
0	1	0	Deferred. The response agent will service the request later.
0	1	1	Reserved
1	0	0	Hard Failure. The response agent cannot service the request.
1	0	1	No Data. No data is needed from the response agent, as in a write for example.
1	1	0	Implicit Writeback. The transaction resulted in a hit on a modified line, and the snoop agent must supply the data. The response agent receives the writeback.
1	1	1	Normal Data. Data will be sent by the response agent, as in a normal read.

The response agent asserts TRDY when it is ready to accept the written data or writeback data.

3.2.8.6 Data Phase

The data pins are:

- Data Bus Ready (DRDY#)
- Data Bus Busy (DBSY#)
- Data Bus (D[63-0]#).

The driving agent on the bus asserts DRDY# to indicate that the data being driven onto the data bus is valid. The bus driver must assert DRDY# for each bus clock in which there is valid data to be transferred. DRDY# can be deasserted to insert wait states between data transfers.

DBSY# is used to hold the bus during wait states before the first DRDY# and between consecutive DRDY#. DBSY# indicates to other bus agent that the data bus is in use even though there is no valid data currently on the bus.

DBSY# need not be asserted for single clock data transfers with no wait states.

3.2.9 Interrupt Control

The interrupt control signals (INTR, NMI, SMI#) allow the execution of the current instruction stream to be interrupted and suspended.

Maskable Interrupt Request (INTR) is an active high level-sensitive input which causes the processor to suspend execution of the current instruction stream and begin execution of an interrupt service routine. The INTR input can be masked (ignored) through the IF bit in the Flags Register.

When not masked, the Cyrix III CPU responds to the INTR input by performing an interrupt acknowledge bus cycle. During the interrupt acknowledge cycle, the Cyrix III CPU reads the interrupt vector (an 8-bit value), from the data bus. The 8-bit interrupt vector indicates the interrupt level that caused generation of the

INTR and is used by the CPU to determine the beginning address of the interrupt service routine. To assure recognition of the INTR request, INTR must remain active until the start of the interrupt acknowledge cycle.

Non-Maskable Interrupt Request (NMI) is a rising edge sensitive input which causes the processor to suspend execution of the current instruction stream and begin execution of an NMI interrupt service routine. The NMI interrupt cannot be masked by the IF bit in the Flags Register. Asserting NMI causes an interrupt which internally supplies interrupt vector 2h to the CPU core. Therefore, external interrupt acknowledge cycles are not issued.

Once NMI processing has started, no additional NMIs are processed until an IRET instruction is executed, typically at the end of the NMI service routine. If NMI is re-asserted prior to execution of the IRET, one and only one NMI rising edge is stored and then processed after execution of the next IRET.

System Management Interrupt Request (SMI#) is an interrupt input with higher priority than the NMI input. Asserting SMI# forces the processor to save the CPU state to SMM memory and to begin execution of the SMI service routine.

SMI# behaves one of two ways depending on the SMM mode of the CPU.

In SL-compatible mode SMI# is a falling edge sensitive input and is sampled on every rising edge of the processor input clock. Once SMI# servicing has started, no additional SMI# interrupts are processed until a RSM instruction is executed. If SMI# is reasserted prior to execution of a RSM instruction, one and only one SMI# falling edge is stored and then processed after execution of the next RSM.

In Cyrix enhanced SMM mode, SMI# is level sensitive. As a level sensitive input, software can process all SMI interrupts until all sources in the chipset have cleared.

3.2.10 FPU Error Interface

The FPU interface signals FERR# and IGNNE# are used to control error reporting for the on-chip floating point unit. These signals are typically used for a PC-compatible system implementation. For other applications, FPU errors are reported to the Cyrix III CPU core through an internal interface.

Floating Point Error Status (FERR#) is an active low output asserted by the Cyrix III CPU when an unmasked floating point error occurs. FERR# is asserted during execution of the FPU instruction that caused the error.

Ignore Numeric Error (IGNNE#) is an active low input which forces the Cyrix III CPU to ignore any pending unmasked FPU errors and allows continued execution of floating point instructions. When IGNNE# is not asserted and an unmasked FPU error is pending, the Cyrix III CPU only executes the following floating point instructions: FNCLEX, FNINIT, FNSAVE, FNSTCW, FNSTENV, and FNSTSW#. IGNNE# is ignored when the NE bit in CR0 is set to a 1.

4 ELECTRICAL SPECIFICATIONS

4.1 Introduction

This chapter describes the electrical interface of the Cyrix III processor and provides AC and DC specifications.

4.2 Electrical Ground

All voltage values in Electrical Specifications are measured with respect to Vss ground (GND pins) unless otherwise noted.

4.3 Power Supply Voltage Signalling

The Cyrix III CPU operates using one power supply voltage (Vcc) typically at 2.2 volts, as determined by the VID bus. Hard-wired within the CPU, the 5-pin VID bus signals its core voltage requirement to the motherboard regulator.

4.4 Power and Ground Connections

The Cyrix III CPU contains 370 pins including 86 power pins and 80 ground (GND) pins. The power pins are divided into 73 Vcc pins, eight VREF pins and one Vcc_CMOS pin. The Vcc supply core voltage, the VREF pins are use to establish reference voltage for GTL+ logic (see below) and the Vcc_CMOS pin supplies I/O voltage to a portion of the I/O interface.

4.4.1 Decoupling

Testing and operating the Cyrix III CPU requires the use of standard high frequency techniques to reduce parasitic effects. The high clock frequencies used in the Cyrix III CPU and its output buffer circuits can cause transient power surges when several output buffers switch output levels simultaneously. These effects can be minimized by filtering the DC power leads with low-inductance decoupling capacitors, using low impedance wiring, and by utilizing all of the V_{CC} and GND pins.

4.5 Gunning Transceiver Logic

Many of the I/O interface signals in the Cyrix III and Celeron processors use a variation of the Gunning Transceiver Logic (GTL+) to help eliminate ringing and signal degradations caused by the fast switching.

The GTL+ logic uses a reference level voltage to determine switch point between a logical one and zero. The reference level voltage (VREF) is split into eight individual sources for individual decoupling and appears on eight VREF[7-0] pins to eliminate significant cross coupling.

4.5.1 Pull-Up/Pull-Down Resistors

Table 4-1 lists the input pins that are internally connected to pull-up and pull-down resistors. The pull-up resistors are connected to V_{CC} and the pull-down resistors are connected to ground (GND). When unused, these inputs do not require connection to external pull-up or pull-down resistors.

Table 4-1. Pins Connected to Internal Pull-Up Resistors

SIGNAL	PIN NO.	RESISTOR
BSEL1	AK30	20-kΩ down
BSEL0	AJ33	20-kΩ pull-up
TCK	AL33	20-kΩ pull-up
TDI	AN35	20-kΩ pull-up
THER-MTRIP#	AH28	20-kΩ pull-up
TMS	AK32	20-kΩ pull-up
TRST#	AN33	20-kΩ pull-up
EXTRATPIN#	AD2	20-kΩ pull-up-

4.5.2 NC Connection and Reserved Pins

Pins in the Cyrix III processor that are functional in the Celeron are left disconnected in the Cyrix III processor and designated as NC pins. These pins can be used as needed by the motherboard designer.

Reserved (RESV) pins are used by Cyrix for factory testing or for future use. These pins should not be connected to any component on the motherboard. Connecting a reserved pin to a pull-up resistor, pull-down resistor, or an active signal could cause unexpected results and possible circuit malfunctions.

4.5.3 Absolute MaximumRatings

The following table lists absolute maximum ratings for the Cyrix III CPU processors. Stresses beyond those listed under Table 4-2 limits may cause permanent damage to the device. These are stress ratings only and do not imply that operation under any conditions other than those listed under “Recommended Operating Conditions” Table 4-3 (Page 4-136) is possible. Exposure to conditions beyond Table 4-2 may (1) reduce device reliability and (2) result in premature failure even when there is no immediately apparent sign of failure. Prolonged exposure to conditions at or near the absolute maximum ratings may also result in reduced useful life and reliability.

Table 4-2. Absolute Maximum Ratings

PARAMETER	MIN	MAX	UNITS	NOTES
Operating Case Temperature	5	85	°C	
Storage Temperature	-40	85	°C	
Max VID pin current		5	ma	

Notes:

1. Operating voltage is the voltage to which the component id designed to operate.
2. This rating applies to Vcc, and any input (except noted below) to the processor.
3. Parameter applies to CMOS and JTAG bus signal groups only.

4.6 Recommended Operating Conditions

Table 4-3 presents the recommended operating conditions for the Cyrix III CPU device.

Table 4-3. Recommended Operating Conditions for CMOS Signals

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS	NOTES
T_C	Operating Case Temperature	0		70	°C	Power Applied
$V_{CC\ CORE}$	Core Supply Voltage (2.2 V)	2.1		2.3	V	
$V_{IL\ CMOS}$	CMOS Input Low Voltage	-0.3		0.7	V	
$V_{IH\ CMOS}$	CMOS Input High Voltage	1.7		2.625	V	
$I_{OL\ CMOS}$	Output Low Current	14			mA	

Table 4-4. Recommended Operating Conditions for GTL+ Signals

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS	NOTES
$V_{IL\ GTL+}$	GTL+ Input Low Voltage	-0.3		0.82	V	
$V_{IH\ GTL+}$	GTL+ Input High Voltage	1.22		V_{TT}	V	
$I_{OL\ GTL+}$	GTL+ Output Low Current	36		48	mA	
$V_{TT\ GTL+}$	GTL+ Bus Termination Voltage	1.365	1.5	1.635	V	
$B_{TT\ GTL+}$	GTL+ Bus Termination Resistance		56		Ohms	
$V_{REF\ GTL+}$	GTL+ Input Reference Voltage	2/3 V_{TT} –2%	2/3 V_{TT}	2/3 V_{TT} +2%	V	

4.7 Bus Signal Groups

The Cyrix III bus can be divided into four bus-signals groups. These groups are listed in Table 4-5 below.

Table 4-5. Bus Signal Groups

SIGNAL TYPE	PARAMETER
GTL Input	BPRI#, DEFER#, RESET#, RS[2:0]#, TRDY#
GTL Input/Output	A[31:3]#, ADS#, BNR#, D[63:0]#, DBSY#, DRDY#, HIT#, HITM#, LOCK#, REQ[4:0]#
CMOS Input	A20M#, BSEL0, BSEL1, CLK,FLUSH#, IGNNE#, INIT#, INTR, NMI, PWRGOOD, SMI#, SLP#, STPCLK#
CMOS Output	FERR#
Continuous DC Level	CPUPRES#, VID[4:0]

4.8 DC Characteristics

Table 4-6. DC Characteristics for CMOS Signals (at Recommended Operating Conditions)

SYMBOL	PARAMETER	MAX	UNIT	NOTES
V_{OL}	Output Low Voltage	0.4	V	
V_{OH}	Output High Voltage	2.625	V	
I_L	Leakage Current	± 100	uA	
I_{LO}	Output Leakage Current	± 10	uA	

Table 4-7. DC Characteristics for GTL+ Signals (at Recommended Operating Conditions)

SYMBOL	PARAMETER	MAX	UNITS	NOTES
V_{OL}	Output Low Voltage	0.60	V	Measured into 25 ohm resistor to 1.5v
V_{OH}	Output High Voltage	V_{TT}	V	See bus termination table
I_L	Leakage Current	± 100	uA	
I_{LO}	Output Leakage Current	± 15	uA	

Table 4-8. DC Characteristics (at Recommended Operating Conditions)

PARAMETER	ICC CORE MAX	UNITS	NOTES
I_{CC} Active I_{CC}			Notes 1, 2
333 MHz	9.15	A	
366 MHz	9.76		
400 MHz	10.40		
433 MHz	10.85		
450 MHz	11.20		
$I_{CC\ SG}$ Stop Grant I_{CC}			Notes 1, 2, 3
PR 433 (333 MHz)	1.05	A	
PR 466 (366 MHz)	1.10		
PR 500 (400 MHz)	1.15		
PR 533 (433 MHz)	1.20		
PR 533 (450 MHz)	1.25		
$I_{CC\ SM}$ Sleep Mode I_{CC}			
PR 433 (333 MHz)	0.87	A	Notes 1, 2, 4
PR 466 (366 MHz)	0.89		
PR 500 (400 MHz)	0.91		
PR 533 (433 MHz)	0.93		
PR 533 (450 MHz)	0.95		

Notes:
1. These values should be used for power supply design. Maximum I_{CC} is determined using the worst-case instruction sequences and functions at maximum Vcc.
2. Frequency (MHz) ratings refer to the internal clock frequency.
3. All inputs at 0.4 or V_{CC} - 0.4 (CMOS levels). All inputs held static except clock and all outputs unloaded (static I_{OUT} = 0 mA).
4. All inputs at 0.4 or V_{CC} - 0.4 (CMOS levels). All inputs held static and all outputs unloaded (static I_{OUT} = 0 mA).

Table 4-9. Power Dissipation

PARAMETER	POWER	UNITS	NOTES
P_{CC} Active Power Dissipation			Note 1
PR 433 (333 MHz)	20.1	W	
PR 466 (366 MHz)	21.5		
PR 500 (400 MHz)	22.9		
PR 533 (433 MHz)	23.9		
PR 533 (450 MHz)	24.6		
P_{SG} Stop Grant Power Dissipation			Notes 1, 2
PR 433 (333 MHz)	2.31	W	
PR 466 (366 MHz)	2.42		
PR 500 (400 MHz)	2.53		
PR 533 (433 MHz)	2.64		
PR 533 (450 MHz)	2.75		
P_{SM} Sleep Mode Power Dissipation			Notes 1, 3
PR 433 (333 MHz)	1.91	W	
PR 466 (366 MHz)	1.96		
PR 500 (400 MHz)	2.00		
PR 533 (433 MHz)	2.05		
PR 533 (450 MHz)	2.09		

Notes:
1. Systems must be designed to thermally dissipate the maximum active power dissipation. Maximum power is determined using the worst-case instruction sequences and functions at maximum V_{CC}.
2. All inputs at 0.4 or V_{CC} - 0.4 (CMOS levels). All inputs held static except clock and all outputs unloaded (static I_{OUT} = 0 mA).
3. All inputs at 0.4 or V_{CC} - 0.4 (CMOS levels). All inputs held static and all outputs unloaded (static I_{OUT} = 0 mA).

4.9 AC Characteristics

The preliminary AC characteristics for the system bus clock, BCLK, and the Cyrix III GTL and CMOS signals at different bus clock speeds are listed in the tables below.

Table 4-10. 66 MHz System Bus AC Characteristics

SYMBOL	PARAMETER	MIN	TYPICAL	MAX	UNIT	NOTES
BCLK	System Bus Frequency		66.67		MHz	
T1	BCLK Period		15.0		ns	
T2	BCLK Period Stability			± 300	ps	
T3	BCLK High Time	3.6			ns	
T4	BCLK Low Time	3.6			ns	
T5	BCLK Rise Time	0.34		1.40	ns	0.5v to 2.0v
T6	BCLK Fall Time	0.34		1.40	ns	2.0v to 0.5v

Table 4-11. 100 MHz System Bus AC Characteristics

SYMBOL	PARAMETER	MIN	TYPICAL	MAX	UNIT	NOTES
BCLK	System Bus Frequency		100.00		MHz	
T1	BCLK Period		10.0		ns	
T2	BCLK Period Stability			± 250	ps	
T3	BCLK High Time	2.4			ns	
T4	BCLK Low Time	2.4			ns	
T5	BCLK Rise Time	0.34		1.40	ns	
T6	BCLK Fall Time	0.34		1.40	ns	

Table 4-12. 133 MHz System Bus AC Characteristics

SYMBOL	PARAMETER	MIN	TYPICAL	MAX	UNIT	NOTES
BCLK	System Bus Frequency		133.00		MHz	
T1	BCLK Period		7.5		ns	
T2	BCLK Period Stability			± 200	ps	
T3	BCLK High Time	1.8			ns	
T4	BCLK Low Time	1.8			ns	
T5	BCLK Rise Time	0.34		1.40	ns	
T6	BCLK Fall Time	0.34		1.40	ns	

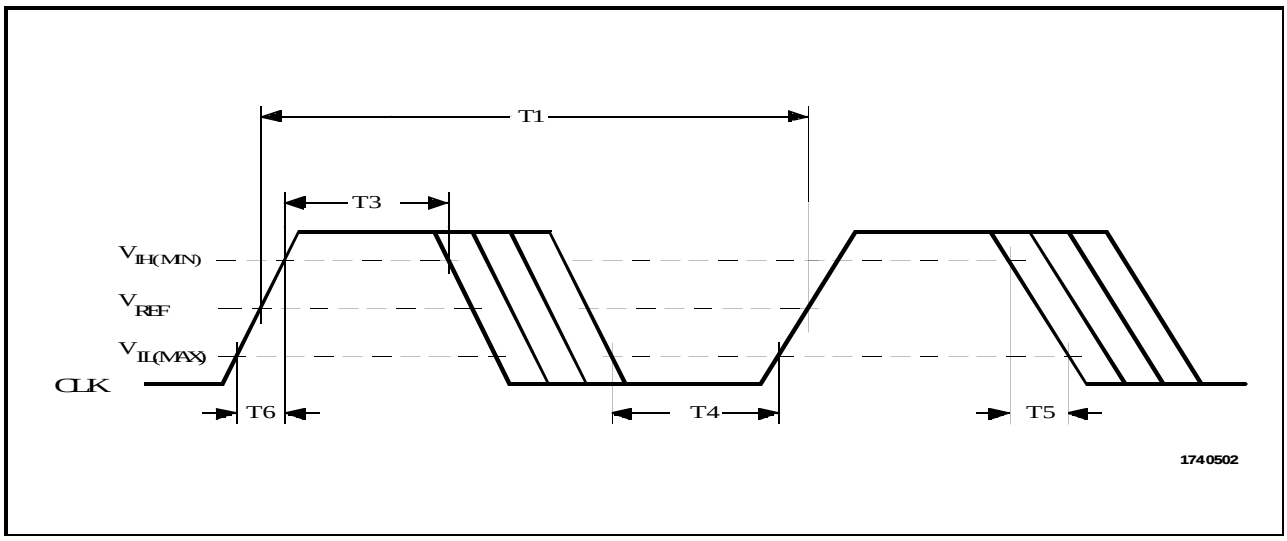


Figure 4-2. BCLK Timing and Measurement Points

Table 4-13. GTL+ Signal AC Characteristics

PARAMETER	66 MHz		100 MHz		133 MHz		UNIT	NOTES
	MIN	MAX	MIN	MAX	MIN	MAX		
GTL+ Output Valid Delay	0.17	4.40	0.17	3.45	0.17	3.00	ns	1,2
GTL+ Input Setup Time	1.60		1.60		1.40		ns	1
GTL+ Input Hold Time	0.90		0.90		0.90		ns	1
RESET Pulse Width	1.00		1.00		1.00		ms	3

Note:
1. All timings are referenced from the rising edge of BCLK at 1.25V and are measured to the GTL+ signal when it crosses 1.00 Volt.
2. Valid delay timings are specified for a 25 ohm resistance to V_{TT} and with V_{REF} at 1.0 Volt.
3. RESET# must remain asserted for the time specified after $V_{CC\ CORE}$ and BCLK are stable

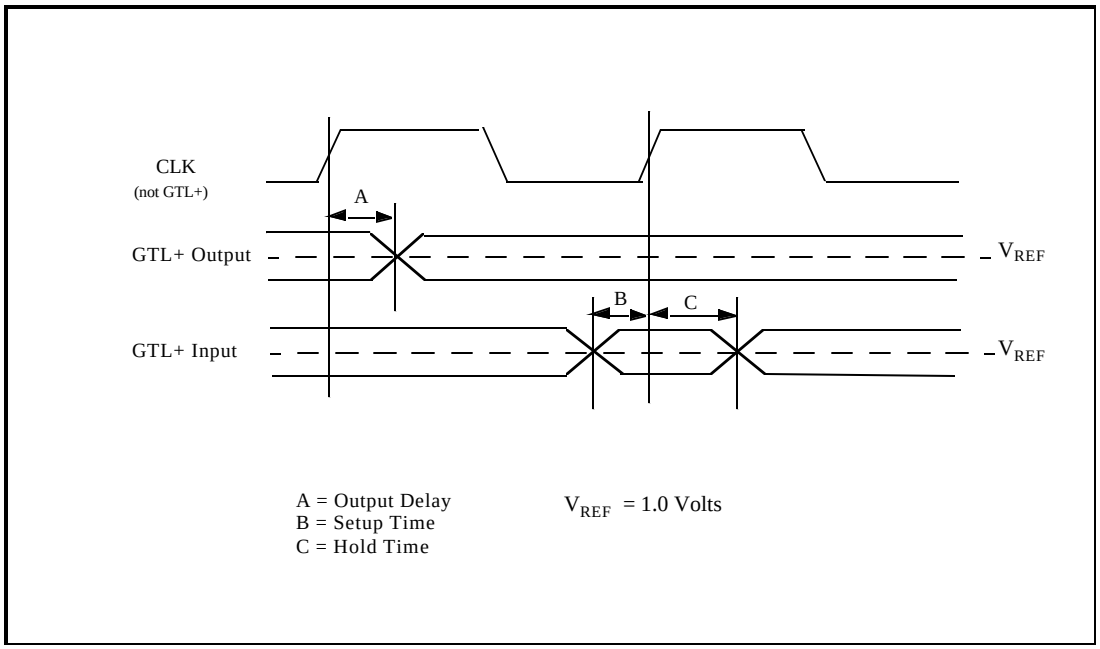


Figure 4-3. GTL+ Signal Definition

Table 4-14. CMOS Signal AC Characteristics

PARAMETER	66 MHz		100 MHz		133 MHz		UNIT	NOTES
	MIN	MAX	MIN	MAX	MIN	MAX		
2.5 V Output Valid Delay	0	8.0	0	8.0	0	7.0	ns	
2.5 V Input Setup Time	4.0		4.0		4.0		ns	See below
2.5 V Input Hold Time	1.3		1.3		1.3		ns	

Note. All timings are referenced from the rising edge of BCLK at 1.25V and are measured to the CMOS signal when it crosses 1.25 V.olts.

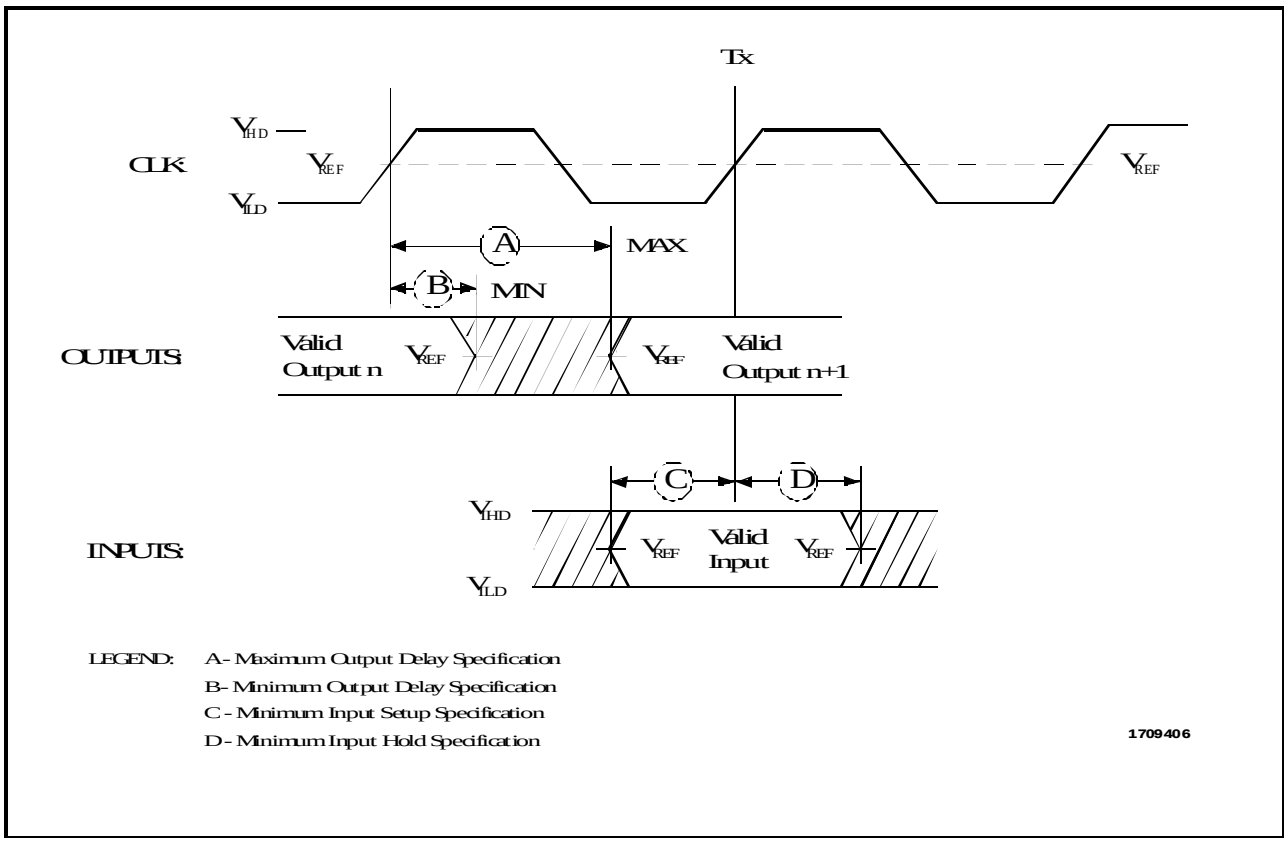


Figure 4-4. Drive Level and Measurement Points for Switching Characteristics

5 MECHANICAL SPECIFICATIONS

5.1 370-Pin SPGA Package

The pin assignments for the Cyrix III CPU in a 370-pin SPGA package are shown in Figure 5-1. Pin lists and dimensions are also included in this chapter.

Figure 5-1. Bottom View 370-Pin SPGA Package Pin Assignments

370-Pin SPGA Package

Figure 5-5. Top View 370-Pin SPGA Package Pin Assignments

Note: REVS = Reserved

Table 5-15. 370-Pin SPGA Package Signal Names Sorted by Signal Name

Signal Name	Pin	Type	I/O
A3#	AK8	GTLP	I/O
A4#	AH12	GTLP	I/O
A5#	AH8	GTLP	I/O
A6#	AN9	GTLP	I/O
A7#	AL15	GTLP	I/O
A8#	AH10	GTLP	I/O
A9#	AL9	GTLP	I/O
A10#	AH6	GTLP	I/O
A11#	AK10	GTLP	I/O
A12#	AN5	GTLP	I/O
A13#	AL7	GTLP	I/O
A14#	AK14	GTLP	I/O
A15#	AL5	GTLP	I/O
A16#	AN7	GTLP	I/O
A17#	AE1	GTLP	I/O
A18#	Z6	GTLP	I/O
A19#	AG3	GTLP	I/O
A20#	AC3	GTLP	I/O
A21#	AJ1	GTLP	I/O
A22#	AE3	GTLP	I/O
A23#	AB6	GTLP	I/O
A24#	AB4	GTLP	I/O
A25#	AF6	GTLP	I/O
A26#	Y3	GTLP	I/O
A27#	AA1	GTLP	I/O
A28#	AK6	GTLP	I/O
A29#	Z4	GTLP	I/O
A30#	AA3	GTLP	I/O
A31#	AD4	GTLP	I/O
A20M#	AE33	CMOS	I
ADS#	AN31	GTLP	I/O
BCLK	W37	GTLP	I
BNR#	AH14	GTLP	I/O
BPRI#	AN17	GTLP	I
BR0#	AN29	Ground	O
BSEL0	AJ33	CMOS/PU	I
BSEL1	AK30 AJ31	CMOS/PD	I
CPUPRES#	C37	GND	O
D0#	W1	GTLP	I/O
D1#	T4	GTLP	I/O
D2#	N1	GTLP	I/O
D3#	M6	GTLP	I/O
D4#	U1	GTLP	I/O
D5#	S3	GTLP	I/O
D6#	T6	GTLP	I/O
D7#	J1	GTLP	I/O
D8#	S1	GTLP	I/O
D9#	P6	GTLP	I/O
D10#	Q3	GTLP	I/O
D11#	M4	GTLP	I/O
D12#	Q1	GTLP	I/O
D13#	L1	GTLP	I/O
D14#	N3	GTLP	I/O
D15#	U3	GTLP	I/O
D16#	H4	GTLP	I/O
D17#	R4	GTLP	I/O
D18#	P4	GTLP	I/O
D19#	H6	GTLP	I/O
D20#	L3	GTLP	I/O
D21#	G1	GTLP	I/O
D22#	F8	GTLP	I/O
D23#	G3	GTLP	I/O
D24#	K6	GTLP	I/O
D25#	E3	GTLP	I/O
D26#	E1	GTLP	I/O
D27#	F12	GTLP	I/O
D28#	A5	GTLP	I/O
D29#	A3	GTLP	I/O
D30#	J3	GTLP	I/O
D31#	C5	GTLP	I/O
D32#	F6	GTLP	I/O
D33#	C1	GTLP	I/O
D34#	C7	GTLP	I/O
D35#	B2	GTLP	I/O
D36#	C9	GTLP	I/O
D37#	A9	GTLP	I/O
D38#	D8	GTLP	I/O
D39#	D10	GTLP	I/O
D40#	C15	GTLP	I/O
D41#	D14	GTLP	I/O
D42#	D12	GTLP	I/O
D43#	A7	GTLP	I/O
D44#	A11	GTLP	I/O
D45#	C11	GTLP	I/O
D46#	A21	GTLP	I/O
D47#	A15	GTLP	I/O
D48#	A17	GTLP	I/O
D49#	C13	GTLP	I/O
D50#	C25	GTLP	I/O
D51#	A13	GTLP	I/O
D52#	D16	GTLP	I/O
D53#	A23	GTLP	I/O
D54#	C21	GTLP	I/O
D55#	C19	GTLP	I/O
D56#	C27	GTLP	I/O
D57#	A19	GTLP	I/O
D58#	C23	GTLP	I/O
D59#	C17	GTLP	I/O
D60#	A25	GTLP	I/O
D61#	A27	GTLP	I/O
D62#	E25	GTLP	I/O
D63#	F16	GTLP	I/O
DBSY#	AL27	GTLP	I/O
DEFER#	AN19	GTLP	I/O
DRDY#	AN27	GTLP	I/O
EXTRATIO#	AB2	CMOS/PD	O
FERR#	AC35	CMOS	O
FLUSH#	AE37	CMOS	I
HIT#	AL25	GTLP	I/O
HITM#	AL23	GTLP	I/O
IERR#	AE35	Pull Up	O
IGNNE#	AG37	CMOS	I
INIT#	AG33	CMOS	I
INTR	M36	CMOS	I
LOCK#	AK20	GTLP	I/O
NMI	L37	CMOS	I
PLL_GND	U33	VSSPLL	
PLL_VDD	W33	VDDPLL	
PWRGOOD	AK26	CMOS	I
REQ0#	AK18	GTLP	I/O
REQ1#	AH16	GTLP	I/O
REQ2#	AH18	GTLP	I/O
REQ3#	AL19	GTLP	I/O
REQ4#	AL17	GTLP	I/O
RESET#	X4	GTLP	I
RS0#	AH26	GTLP	I/O
RS1#	AH22	GTLP	I/O
RS2#	AK28	GTLP	I/O
SLP#	AH30	CMOS	I
SMI#	AJ35	CMOS	I
STPCLK#	AG35	CMOS	I
TCLK	AL33	CMOS/PU	I
TDI	AN35	CMOS/PU	I
TDO	AN37	CMOS	O
THERMDN	AL29	ESD only	O
THERMDP	AL31	ESD only	O
THRMTRIP#	AH28	CMOS/PU	I/O
TMS	AK32	CMOS/PU	I
TRDY#	AN25	GTLP	I/O
TRST#	AN33	CMOS/PU	I
VID0	AL35	Voltage ID	O
VID1	AM36	Voltage ID	O
VID2	AL37	Voltage ID	O

Table 5-2. 370-Pin SPGA Package Signal Names Sorted by Pin Number

Package Pin	Signal Name	Pad Type	I/O
A3	D29#	GTLP	I/O
A5	D28#	GTLP	I/O
A7	D43#	GTLP	I/O
A9	D37#	GTLP	I/O
A11	D44#	GTLP	I/O
A13	D51#	GTLP	I/O
A15	D47#	GTLP	I/O
A17	D48#	GTLP	I/O
A19	D57#	GTLP	I/O
A21	D46#	GTLP	I/O
A23	D53#	GTLP	I/O
A25	D60#	GTLP	I/O
A27	D61#	GTLP	I/O
B2	D35#	GTLP	I/O
C1	D33#	GTLP	I/O
C5	D31#	GTLP	I/O
C7	D34#	GTLP	I/O
C9	D36#	GTLP	I/O
C11	D45#	GTLP	I/O
C13	D49#	GTLP	I/O
C15	D40#	GTLP	I/O
C17	D59#	GTLP	I/O
C19	D55#	GTLP	I/O
C21	D54#	GTLP	I/O
C23	D58#	GTLP	I/O
C25	D50#	GTLP	I/O
C27	D56#	GTLP	I/O
C37	CPUPRES#	GND	O
D8	D38#	GTLP	I/O
D10	D39#	GTLP	I/O
D12	D42#	GTLP	I/O
D14	D41#	GTLP	I/O
D16	D52#	GTLP	I/O
E1	D26#	GTLP	I/O
E3	D25#	GTLP	I/O
E25	D62#	GTLP	I/O
F6	D32#	GTLP	I/O
F8	D22#	GTLP	I/O
F12	D27#	GTLP	I/O
F16	D63#	GTLP	I/O
G1	D21#	GTLP	I/O
G3	D23#	GTLP	I/O
H4	D16#	GTLP	I/O
H6	D19#	GTLP	I/O
J1	D7#	GTLP	I/O
J3	D30#	GTLP	I/O
K6	D24#	GTLP	I/O
L1	D13#	GTLP	I/O
L3	D20#	GTLP	I/O
L37	NMI	CMOS	I
M4	D11#	GTLP	I/O
M6	D3#	GTLP	I/O
M36	INTR	CMOS	I
N1	D2#	GTLP	I/O
N3	D14#	GTLP	I/O
P4	D18#	GTLP	I/O
P6	D9#	GTLP	I/O
Q1	D12#	GTLP	I/O
Q3	D10#	GTLP	I/O
R4	D17#	GTLP	I/O
S1	D8#	GTLP	I/O
S3	D5#	GTLP	I/O
T4	D1#	GTLP	I/O
T6	D6#	GTLP	I/O
U1	D4#	GTLP	I/O
U3	D15#	GTLP	I/O
U33	PLL_GND	VSSPLL	
W1	D0#	GTLP	I/O
W33	PLL_VDD	VDDPLL	
W37	BCLK	GTLP	I
X4	RESET#	GTLP	I
Y3	A26#	GTLP	I/O
Z4	A29#	GTLP	I/O
Z6	A18#	GTLP	I/O
AA1	A27#	GTLP	I/O
AA3	A30#	GTLP	I/O
AB2	EXTRATIO#	CMOS/PD	O
AB4	A24#	GTLP	I/O
AB6	A23#	GTLP	I/O
AC3	A20#	GTLP	I/O
AC35	FERR#	CMOS	O
AD4	A31#	GTLP	I/O
AE1	A17#	GTLP	I/O
AE3	A22#	GTLP	I/O
AE33	A20M#	CMOS	I
AE35	IERR#	Pull Up	O
AE37	FLUSH#	CMOS	I
AF6	A25#	GTLP	I/O
AG3	A19#	GTLP	I/O
AG33	INIT#	CMOS	I
AG35	STPCLK#	CMOS	I
AG37	IGNNE#	CMOS	I
AH10	A8#	GTLP	I/O
AH12	A4#	GTLP	I/O
AH14	BNR#	GTLP	I/O
AH16	REQ1#	GTLP	I/O
AH18	REQ2#	GTLP	I/O
AH22	RS1#	GTLP	I/O
AH26	RS0#	GTLP	I/O
AH28	THRMTRIP#	CMOS/PU	I/O
AH30	SLP#	CMOS	I
AH6	A10#	GTLP	I/O
AH8	A5#	GTLP	I/O
AJ1	A21#	GTLP	I/O
AJ31	BSEL1	CMOS/PD	
AJ33	BSEL0	CMOS/PU	I
AJ35	SMI#	CMOS	I
AJ37	VID3	Voltage ID	O
AK6	A28#	GTLP	I/O
AK8	A3#	GTLP	I/O
AK10	A11#	GTLP	I/O
AK14	A14#	GTLP	I/O
AK18	REQ0#	GTLP	I/O
AK20	LOCK#	GTLP	I/O
AK26	PWRGOOD	CMOS	I
AK28	RS2#	GTLP	I/O
AK30	BSEL1	CMOS/PD	I
AK32	TMS	CMOS/PU	I
AK36	VID4	Voltage ID	O
AL5	A15#	GTLP	I/O
AL7	A13#	GTLP	I/O
AL9	A9#	GTLP	I/O
AL15	A7#	GTLP	I/O
AL17	REQ4#	GTLP	I/O
AL19	REQ3#	GTLP	I/O
AL23	HITM#	GTLP	I/O
AL25	HIT#	GTLP	I/O
AL27	DBSY#	GTLP	I/O
AL29	THERMDN	ESD only	O
AL31	THERMDP	ESD only	O
AL33	TCLK	CMOS/PU	I
AL35	VID0	Voltage ID	O
AL37	VID2	Voltage ID	O
AM36	VID1	Voltage ID	O
AN5	A12#	GTLP	I/O
AN7	A16#	GTLP	I/O
AN9	A6#	GTLP	I/O
AN17	BPRI#	GTLP	I
AN19	DEFER#	GTLP	I/O
AN25	TRDY#	GTLP	I/O
AN27	DRDY#	GTLP	I/O

Table 5-15. Voltage and Reserved Pins

VOLTAGE LEVEL	PURPOSE	PINS
VCC	Voltage Inputs. Supplies nominal 2.2 volts to processor core.	B6, B10, B14, B18, B22, B26, B30, B34, C3, D6, D20, D24, D28, D32, D36, E5, E9, E13, E17, H32, F2, F4, F14, F22, F26, F30, F34, H36, J5, K2, K32, K34, M32, N5, P2, P34, R32, R36, S5, T2, T34, V32, V36, W5, X34, Y35, Z32, AA5, AA37, AB2, AB34, AD32, AE5, AF2, AF34, AH24, AH32, AH36, AJ5, AJ9, AJ13, AJ17, AJ21, AJ25, AJ29, AK2, AK34, AM12, AM4, AM8, AM16, AM20, AM24, AM28, AM32
VCC_1.5V	Voltage Input. Supplies 1.5 volts to processor core	AD36
VCC_2.5V	Voltage Input. Supplies 2.5 volts to processor I/O	Z36
VCC_CMOS	Voltage Output. Connects to VCC_2.5. Supplies 2.5 volts to motherboard or other component.	AB36
VREF[0-7]	Voltage Input. Establishes reference voltage for GTL+ logic switching level.	V_{REF0} E33 V_{REF1} F18 V_{REF2} K4 V_{REF3} R6 V_{REF4} V6 V_{REF5} AD6 V_{REF6} AK12 V_{REF7} AK22
GND	Return path for all voltages.	A37, B4, B8, B12, B16, B20, B24, B28, B32, D2, D4, D18, D22, D26, D30, D34, E7, E11, E15, E19, F20, F24, F28, F32, F36, G5, H2, H34, K36, L5, M2, M34, P32, P36, Q5, R34, T32, T36, U5, V2, V34, X36, Y5, Y33, Y37, X32, Z2, Z34, AB32, AC33, AC5, AD2, AD34, AF32, AF36, AG5, AH2, AH34, AJ3, AJ7, AJ11, AJ15, AJ19, AJ23, AJ27, AJ31, AK4, AL1, AL3, AM6, AM10, AM14, AM18, AM2, AM22, AM26, AM30, AM34, AN3
Reserved Pins	These pins are used for factory testing or reserved for future use. Generally compatible with Celeron™ processor defined pins.	A29, A31, A33, A35, B36, C29, C31, C33, C35, E21, E23, E27, E29, E31, E35, E37, F10, G33, G35, G37, J33, J37, J35, L33, L35, N33, N35, N37, Q33, Q35, Q37, R2, S33, S35, S37, U35, U37, V4, W3, W35, X2, X6, Y1, AA33, AA35, AC1, AC37, AF4, AG1, AH4, AH20, AK16, AK24, AL11, AL13, AL21, AN11, AN13, AN15, AN21, AN23, AN29

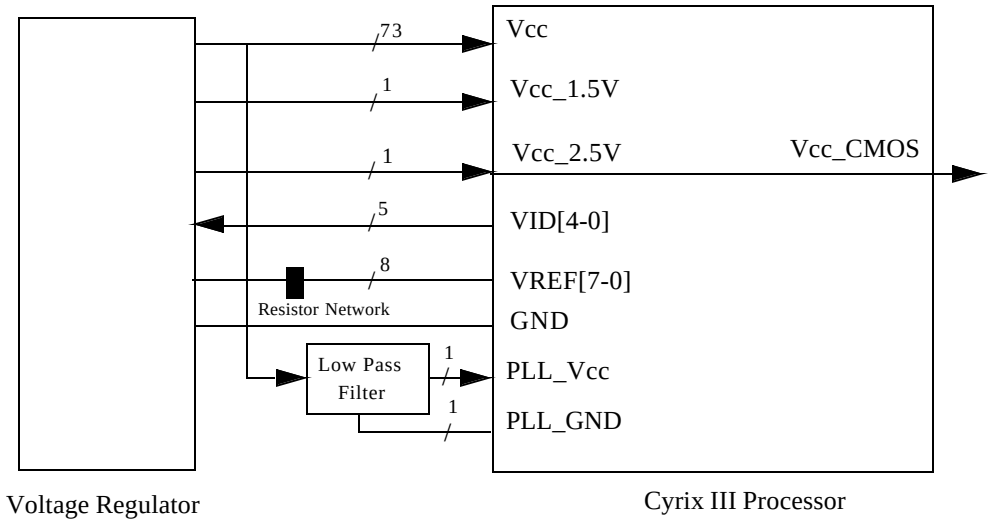


Figure 5-3. Typical Connections Between Voltage Regulator and Cyrix III Processor

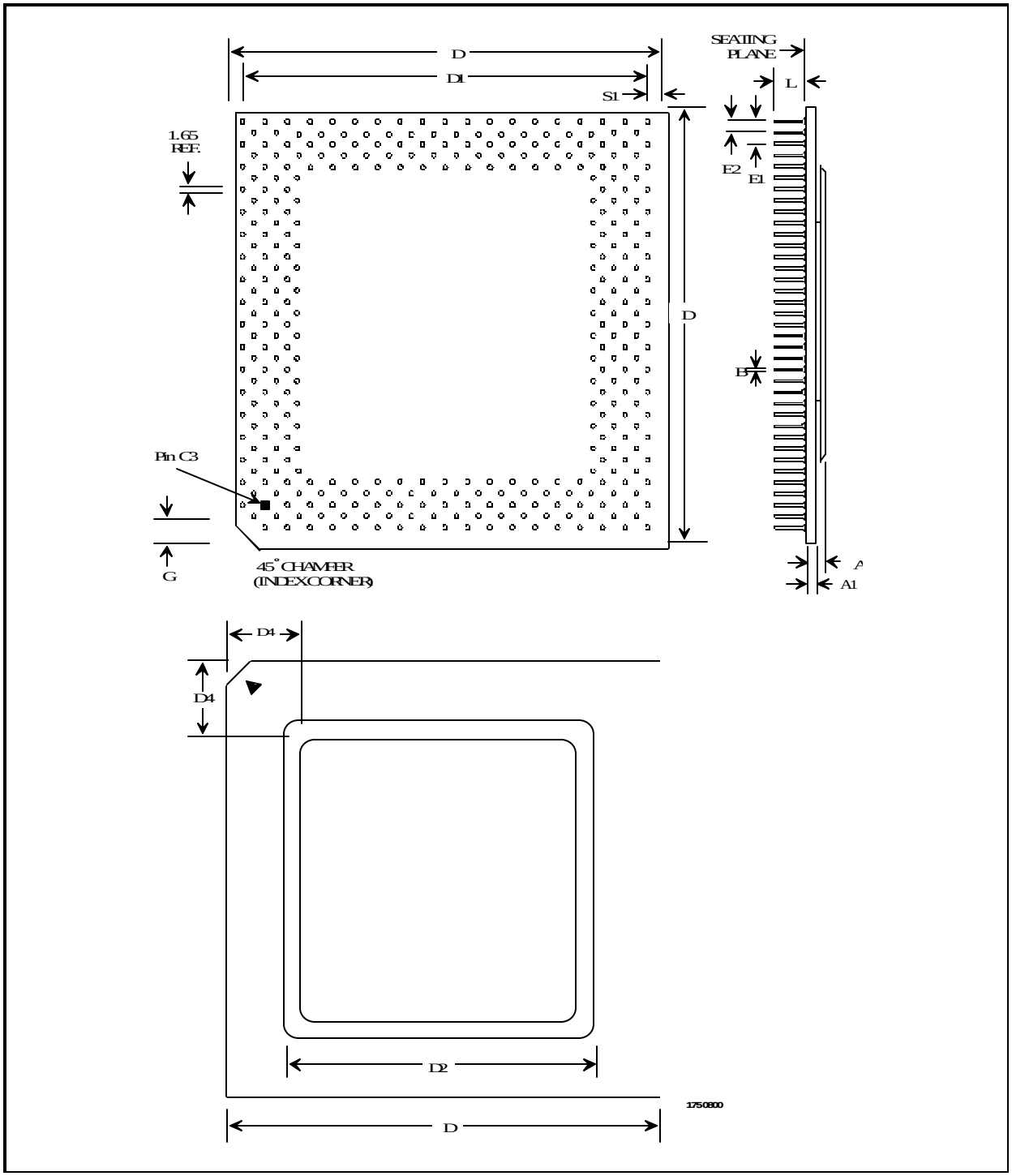


Figure 5-4. 370-Pin “Flip Chip SPGA

Table 5-16. 370-Pin SPGA Dimensions

SYMBOL	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	3.80	4.50	0.150	0.177
A1	1.62	1.98	0.064	0.078
B	0.43	0.51	0.017	0.020
D	49.28	49.91	1.940	1.965
D1	45.47	45.97	1.790	1.810
D2	36.75 Sq.	37.25 Sq.	1.447	1.467
E1	2.41	2.67	0.095	0.105
E2	1.14	1.40	0.045	0.055
G	1.52	2.29	0.060	0.090
L	2.97	3.38	0.117	0.133
S1	1.65	2.16	0.065	0.085

5.2 Thermal Resistances

Three thermal resistances can be used to idealize the heat flow from the junction of the Cyrix III CPU to ambient:

θ_{JC} = thermal resistance from junction to case in °C/W

θ_{CS} = thermal resistance from case to heatsink in °C/W,

θ_{SA} = thermal resistance from heatsink to ambient in °C/W,

$\theta_{CA} = \theta_{CS} + \theta_{SA}$, thermal resistance from case to ambient in °C/W.

$T_C = T_A + P * \theta_{CA}$ (where T_A = ambient temperature and P = power applied to the CPU).

To maintain the case temperature under 85°C during operation θ_{CA} can be reduced by a heat-sink/fan combination. (The heatsink/fan decreases θ_{CA} by a factor of three compared to using a heatsink alone.) The required θ_{CA} to maintain 85°C is shown in Table 5-4. The designer should ensure that adequate air flow is maintained to control the ambient temperature (T_A). A typical θ_{JC} value for the Cyrix III 370-pin PGA-package value is 0.5 °C/W.

Cyrix III Performance Rating	Cyrix III Actual MHz	Max Active Current (A)	Max Active Power (W)	Θ_{CA} For Different Ambient Temperatures				
				25°C	30°C	35°C	40°C	45°C
PR 433	333 MHz	9.15	20.1	2.98	2.74	2.50	2.24	1.99
PR 466	366 MHz	9.76	21.5	2.79	2.55	2.33	2.09	1.86
PR 500	400 MHz	10.40	22.9	2.62	2.40	2.18	1.96	1.75
PR 533	433 MHz	10.85	23.9	2.51	2.30	2.09	1.88	1.67
PR 533	450 MHz	11.20	24.6	2.41	2.24	2.03	1.83	1.63

Required θ_{CA} to Maintain 85°C Case Temperature

Thermal Resistances

6 INSTRUCTION SET

This section summarizes the Cyrix III 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 Cyrix III 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 6-158)
Opcode		Instruction operation	6.2.2 (Page 6-159)
mod	Address Mode Specifier	Used with r/m field to select address mode	6.2.3 (Page 6-161)
reg	General Register Specifier	Uses reg, sreg2 or sreg3 encoding depending on opcode field	6.2.4 (Page 6-162)
r/m	Address Mode Specifier	Used with mod field to select addressing mode.	6.2.3 (Page 6-161)
ss	Scale Factor	Scaled-index address mode	6.2.5 (Page 6-164)
Index		Determines general register to be selected as index register	6.2.6 (Page 6-164)
Base		Determines general register to be selected as base register	6.2.7 (Page 6-165)
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 6-162).

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 6-162)	

Note 1: An “s-i-d” (ss, Index, Base) field is present. Refer to the ss Table 6-13 (Page 6-164), Index Table 6-14 (Page 6-164)
and Base Table 6-15 (Page 6-165).

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

ss 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 6-161), 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 Cyrix III 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 I 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

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

REGISTER	CONTENTS
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 Cyrix III CPU instruction set is presented in three tables: Table 6-21. “Cyrix III CPU Instruction Set Clock Count Summary” on page 6-169, Table 6-23. “Cyrix III FPU Instruction Set Summary” on page 6-187 and the Table 6-25. “Cyrix III Processor MMX Instruction Set Clock Count Summary” on page 6-194. Additional information concerning the FPU Instruction Set is presented on page 6-186, and the Cyrix III MMX instruction set on page 6-193.

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 (Cyrix III CPU with 2x clock) or four clocks (Cyrix III 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 ≤ IOPL \ 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".

Table 6-20. Action of Instruction on Flag

INSTRUCTION TABLE SYMBOL	ACTION
u	Flag is undefined following execution of the instruction.

Table 6-21. Cyrix III 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*		x - - - x x x x x			b	h
Register to Register	1 [00dw] [11 reg r/m]		1	1		
Register to Memory	1 [000w] [mod reg r/m]		1	1		
Memory to Register	1 [001w] [mod reg r/m]		1	1		
Immediate to Register/Memory	8 [00sw] [mod 010 r/m]###		1	1		
Immediate to Accumulator	1 [010w] ###		1	1		
ADD *Integer Add*		x - - - x x x x x			b	h
Register to Register	0 [00dw] [11 reg r/m]		1	1		
Register to Memory	0 [000w] [mod reg r/m]		1	1		
Memory to Register	0 [001w] [mod reg r/m]		1	1		
Immediate to Register/Memory	8 [00sw] [mod 000 r/m]###		1	1		
Immediate to Accumulator	0 [010w] ###		1	1		
AND *Boolean AND*		0 - - - x x u x 0			b	h
Register to Register	2 [00dw] [11 reg r/m]		1	1		
Register to Memory	2 [000w] [mod reg r/m]		1	1		
Memory to Register	2 [001w] [mod reg r/m]		1	1		
Immediate to Register/Memory	8 [00sw] [mod 100 r/m]###		1	1		
Immediate to Accumulator	2 [010w] ###		1	1		
ARPL *Adjust Requested Privilege Level*		- - - - - x - - -		9	a	h
From Register/Memory	63 [mod reg r/m]					
BOUND *Check Array Boundaries*	62 [mod reg r/m]	- - - - - - - - - -			b, e	g,h,j,k,r
If Out of Range (Int 5)			20	20+INT		
If In Range			11	11		
BSF *Scan Bit Forward*		- - - - - x - - -	3	3	b	h
Register, Register/Memory	0F BC [mod reg r/m]					
BSR *Scan Bit Reverse*		- - - - - x - - -	3	3	b	h
Register, Register/Memory	0F BD [mod reg r/m]					
BSWAP *Byte Swap*	0F C[1 reg]	- - - - - - - - -	4	4		

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
\- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined
\+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

Table 6-21. Cyrix III 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
BT *Test Bit*		- - - - - - - - - x			b	h
Register/Memory, Immediate	0F BA [mod 100 r/m]#		2	2		
Register/Memory, Register	0F A3 [mod reg r/m]		5/6	5/6		
BTC *Test Bit and Complement*		- - - - - - - - - x			b	h
Register/Memory, Immediate	0F BA [mod 111 r/m]#		3	3		
Register/Memory, Register	0F BB [mod reg r/m]		5/6	5/6		
BTR *Test Bit and Reset*		- - - - - - - - - x			b	h
Register/Memory, Immediate	0F BA [mod 110 r/m]#		3	3		
Register/Memory, Register	0F B3 [mod reg r/m]		5/6	5/6		
BTS *Test Bit and Set*		- - - - - - - - - x			b	h
Register/Memory	0F BA [mod 101 r/m]		3	3		
Register (short form)	0F AB [mod reg r/m]		5/6	5/6		
CALL *Subroutine Call*		- - - - - - - - - -			b	h,j,k,r
Direct Within Segment	E8 +++		1	1		
Register/Memory Indirect Within Segment	FF [mod 010 r/m]		1/3	1/3		
Direct Intersegment	9A [unsigned full offset,		3	4		
Call Gate to Same Privilege	selector]			15		
Call Gate to Different Privilege No Parameters				26		
				35+2m		
Call Gate to Different Privilege m Par's				110		
16-bit Task to 16-bit TSS				118		
16-bit Task to 32-bit TSS				96		
16-bit Task to V86 Task				112		
32-bit Task to 16-bit TSS				120		
32-bit Task to 32-bit TSS				98		
32-bit Task to V86 Task	FF [mod 011 r/m]		5	8		
Indirect Intersegment				20		
Call Gate to Same Privilege				31		
Call Gate to Different Privilege No Parameters				40+2m		
				114		
Call Gate to Different Privilege Level m Par's				122		
				100		
16-bit Task to 16-bit TSS				116		
16-bit Task to 32-bit TSS				124		
16-bit Task to V86 Task				102		
32-bit Task to 16-bit TSS						
32-bit Task to 32-bit TSS						
32-bit Task to V86 Task						
CBW *Convert Byte to Word*	98	- - - - - - - - - -	3	3		
CDQ *Convert Doubleword to Quadword*	99	- - - - - - - - - -	2	2		
CLC *Clear Carry Flag*	F8	- - - - - - - - - 0	1	1		
CLD *Clear Direction Flag*	FC	- 0 - - - - - - -	7	7		
CLI *Clear Interrupt Flag*	FA	- - 0 - - - - - -	7	7		m

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
\- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined

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

Table 6-21. Cyrix III 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
CLTS *Clear Task Switched Flag*	0F 06	- - - - - - - - -	10	10	c	l
CMC *Complement the Carry Flag*	F5	- - - - - - - - x	2	2		
CMP *Compare Integers*		x - - - x x x x x			b	h
Register to Register	3 [10dw] [11 reg r/m]		1	1		
Register to Memory	3 [101w] [mod reg r/m]		1	1		
Memory to Register	3 [100w] [mod reg r/m]		1	1		
Immediate to Register/Memory	8 [00sw] [mod 111 r/m] ###		1	1		
Immediate to Accumulator	3 [110w] ###		1	1		
CMOVA/CMOVNBE *Move if Above/ Not Below or Equal* Register, Register/Memory	0F 47 [mod reg r/m]	- - - - - - - - -	1	1		r
CMOVBE/CMOVNA *Move if Below or Equal/ Not Above* Register, Register/Memory	0F 46 [mod reg r/m]	- - - - - - - - -	1	1		r
CMOVAE/CMOVNB/CMOVNC/ *Move if Above or Equal/Not Below/Not Carry* Register, Register/Memory	0F 43 [mod reg r/m]	- - - - - - - - -	1	1		r
CMOVB/CMOVC/CMOVNAE *Move if Below/ Carry/Not Above or Equal* Register, Register/Memory	0F 42 [mod reg r/m]	- - - - - - - - -	1	1		r
CMOVE/CMOVZ *Move if Equal/Zero* Register, Register/Memory	0F 44 [mod reg r/m]	- - - - - - - - -	1	1		r
CMOVNE/CMOVNZ *Move if Not Equal/ Not Zero* Register, Register/Memory	0F 45 [mod reg r/m]	- - - - - - - - -	1	1		r
CMOVG/CMOVNLE *Move if Greater/ Not Less or Equal* Register, Register/Memory	0F 4F [mod reg r/m]	- - - - - - - - -	1	1		r
CMOVLE/CMOVNG *Move if Less or Equal/ Not Greater* Register, Register/Memory	0F 4E [mod reg r/m]	- - - - - - - - -	1	1		r
CMOVL/CMOVNGE *Move if Less/ Not Greater or Equal* Register, Register/Memory	0F 4C [mod reg r/m]	- - - - - - - - -	1	1		r

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined
\+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

Table 6-21. Cyrix III 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
CMOVGE/CMOVNL *Move if Greater or Equal/ Not Less* Register, Register/Memory	 0F 4D [mod reg r/m]	- - - - - - - - -	 1	 1		r
CMOVO *Move if Overflow* Register, Register/Memory	 0F 40 [mod reg r/m]	- - - - - - - - -	 1	 1		r
CMOVNO *Move if No Overflow* Register, Register/Memory	 0F 41 [mod reg r/m]	- - - - - - - - -	 1	 1		r
CMOVP/CMOVPE *Move if Parity/Parity Even* Register, Register/Memory	0F 4A [mod reg r/m]	- - - - - - - - -	1	1		r
CMONP/CMOVPO *Move if Not Parity/ Parity Odd* Register, Register/Memory	 0F 4B [mod reg r/m]	- - - - - - - - -	 1	 1		r
CMOVS *Move if Sign* Register, Register/Memory	 0F 48 [mod reg r/m]	- - - - - - - - -	 1	 1		r
CMOVNS *Move if Not Sign* Register, Register/Memory	 0F 49 [mod reg r/m]	- - - - - - - - -	 1	 1		r
CMPS *Compare String*	A [011w]	x - - - x x x x x	5	5	b	h
CMPXCHG *Compare and Exchange* 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 *Compare and Exchange* 8 Bytes	0F C7 [mod 001 r/m]	- - - - - - - - -				
CPUID *CPU Identification*	0F A2	- - - - - - - - -	12	12		
CWD *Convert Word to Doubleword*	99	- - - - - - - - -	2	2		
CWDE *Convert Word to Doubleword Extended*	98	- - - - - - - - -	2	2		
DAA *Decimal Adjust AL after Add*	27	- - - - x x x x x	9	9		
DAS *Decimal Adjust AL after Subtract*	2F	- - - - x x x x x	9	9		
DEC *Decrement by 1* 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

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
– = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined
\+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

Table 6-21. Cyrix III 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
DIV *Unsigned Divide*	F [011w] [mod 110 r/m]	- - - - x x u u -			b,e	e,h
Accumulator by Register/Memory						
Divisor: Byte			13-17	13-17		
Word			13-25	13-25		
Doubleword			13-41	13-41		
ENTER *Enter New Stack Frame*	C8 ##,#	- - - - - - - - - -			b	h
Level = 0			10	10		
Level = 1			13	13		
Level (L) > 1			10+L*3	10+L*3		
HLT Halt	F4	- - - - - - - - - -	5	5		l
IDIV *Integer (Signed) Divide*		- - - - - x x u u -			b,e	e,h
Accumulator by Register/Memory	F [011w] [mod 111 r/m]					
Divisor: Byte			16-20	16-20		
Word			16-28	16-28		
Doubleword			17-45	17-45		
IMUL *Integer (Signed) Multiply*		x - - - x x u u x			b	h
Accumulator by Register/Memory	F [011w] [mod 101 r/m]					
Multiplier: Byte			4	4		
Word			4	4		
Doubleword			10	10		
Register with Register/Memory	0F AF [mod reg r/m]					
Multiplier: Word			4	4		
Doubleword			10	10		
Register/Memory with Immediate to Register2						
Multiplier: Word	6 [10s1] [mod reg r/m] ###		5	5		
Doubleword			11	11		
IN *Input from I/O Port*		- - - - - - - - - -				m
Fixed Port	E [010w] [#]		14	14/28		
Variable Port	E [110w]		14	14/28		
INC *Increment by 1*		x - - - x x x x -			b	h
Register/Memory	F [111w] [mod 000 r/m]		1	1		
Register (short form)	4 [0 reg]		1	1		
INS *Input String from I/O Port*	6 [110w]	- - - - - - - - - -	14	14/28	b	h,m

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
\- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined
\+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

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

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined

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

Table 6-21. Cyrix III 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 *Jump on CX/ECX Zero*	E3 +	- - - - - - - - -	1	1		r
JE/JZ *Jump on Equal/Zero*		- - - - - - - - -				r
8-bit Displacement	74 +		1	1		
Full Displacement	0F 84 +++		1	1		
JL/JNGE *Jump on Less/Not Greater or Equal*	7C +	- - - - - - - - -	1	1		r
8-bit Displacement	0F 8C +++		1	1		
Full Displacement						
JLE/JNG *Jump on Less or Equal/Not Greater*	7E +	- - - - - - - - -	1	1		r
8-bit Displacement	0F 8E +++		1	1		
Full Displacement						
JMP *Unconditional Jump*		- - - - - - - - -			b	h,j,k,r
8-bit Displacement	EB +		1	1		
Full Displacement	E9 +++		1	1		
Register/Memory Indirect Within Segment	FF [mod 100 r/m]		1/3	1/3		
Direct Intersegment	EA [unsigned full offset, selector]		1	4		
Call Gate Same Privilege Level				14		
16-bit Task to 16-bit TSS				110		
16-bit Task to 32-bit TSS				118		
16-bit Task to V86 Task				96		
32-bit Task to 16-bit TSS				112		
32-bit Task to 32-bit TSS				120		
32-bit Task to V86 Task				98		
Indirect Intersegment	FF [mod 101 r/m]		5	7		
Call Gate Same Privilege Level				17		
16-bit Task to 16-bit TSS				113		
16-bit Task to 32-bit TSS				121		
16-bit Task to V86 Task				99		
32-bit Task to 16-bit TSS				115		
32-bit Task to 32-bit TSS				123		
32-bit Task to V86 Task				101		
JNB/JAE/JNC *Jump on Not Below/Above or Equal/Not Carry*		- - - - - - - - -				r
8-bit Displacement	73 +		1	1		
Full Displacement	0F 83 +++		1	1		
JNBE/JA *Jump on Not Below or Equal/Above*	77 +	- - - - - - - - -	1	1		r
8-bit Displacement	0F 87 +++		1	1		
Full Displacement						

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
\- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined

\+ = 8-bit signed displacement

+++ = full signed displacement (16, 32 bits)

Table 6-21. Cyrix III 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
JNE/JNZ *Jump on Not Equal/Not Zero*		- - - - - - - - -				r
8-bit Displacement	75 +		1	1		
Full Displacement	0F 85 +++		1	1		
JNL/JGE *Jump on Not Less/Greater or Equal*		- - - - - - - - -				r
8-bit Displacement	7D +		1	1		
Full Displacement	0F 8D +++		1	1		
JNLE/JG *Jump on Not Less or Equal/Greater*		- - - - - - - - -				r
8-bit Displacement	7F +		1	1		
Full Displacement	0F 8F +++		1	1		
JNO *Jump on Not Overflow*		- - - - - - - - -				r
8-bit Displacement	71 +		1	1		
Full Displacement	0F 81 +++		1	1		
JNP/JPO *Jump on Not Parity/Parity Odd*		- - - - - - - - -				r
8-bit Displacement	7B +		1	1		
Full Displacement	0F 8B +++		1	1		
JNS *Jump on Not Sign*		- - - - - - - - -				r
8-bit Displacement	79 +		1	1		
Full Displacement	0F 89 +++		1	1		
JO *Jump on Overflow*		- - - - - - - - -				r
8-bit Displacement	70 +		1	1		
Full Displacement	0F 80 +++		1	1		
JP/JPE *Jump on Parity/Parity Even*		- - - - - - - - -				r
8-bit Displacement	7A +		1	1		
Full Displacement	0F 8A +++		1	1		
JS *Jump on Sign*		- - - - - - - - -				r
8-bit Displacement	78 +		1	1		
Full Displacement	0F 88 +++		1	1		
LAHF *Load AH with Flags*	9F	- - - - - - - - -	2	2		
LAR *Load Access Rights* From Register/Memory	0F 02 [mod reg r/m]	- - - - - x - - -		8	a	g,h,j,p
LDS *Load Pointer to DS*	C5 [mod reg r/m]	- - - - - - - - -	2	4	b	h,i,j

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined

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

Table 6-21. Cyrix III 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
LEA *Load Effective Address*	8D [mod reg r/m]	- - - - - - - - - -				
No Index Register			1	1		
With Index Register			1	1		
LEAVE *Leave Current Stack Frame*	C9	- - - - - - - - - -	4	4	b	h
LES *Load Pointer to ES*	C4 [mod reg r/m]	- - - - - - - - - -	2	4	b	h,i,j
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*		- - - - - - - - - -			a	g,h,j,l
From Register/Memory	0F 00 [mod 010 r/m]		5	5		
LMSW *Load Machine Status Word*		- - - - - - - - - -			b,c	h,l
From Register/Memory	0F 01 [mod 110 r/m]		13	13		
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*		- - - - - x - - - -			a	g,h,j,p
From Register/Memory	0F 03 [mod reg r/m]			8		
LSS *Load Pointer to SS*	0F B2 [mod reg r/m]	- - - - - - - - - -	2	4	a	h,i,j
LTR *Load Task Register*		- - - - - - - - - -			a	g,h,j,l
From Register/Memory	0F 00 [mod 011 r/m]			7		
MOV *Move Data*		- - - - - - - - - -			b	h,i,j
Register to Register	8 [10dw] [11 reg r/m]		1	1		
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		

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined

\+ = 8-bit signed displacement

+++ = full signed displacement (16, 32 bits)

Table 6-21. Cyrix III 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
MOV *Move to/from Control/Debug/Test Regs*	0F 22 [11 eee reg]	- - - - - - - - - -	20/5/5	20/5/5		l
Register to CR0/CR2/CR3/CR4	0F 20 [11 eee reg]		6	6		
CR0/CR2/CR3/CR4 to Register	0F 23 [11 eee reg]		16	16		
Register to DR0-DR3	0F 21 [11 eee reg]		14	14		
DR0-DR3 to Register	0F 23 [11 eee reg]		16	16		
Register to DR6-DR7	0F 21 [11 eee reg]		14	14		
DR6-DR7 to Register	0F 26 [11 eee reg]		10	10		
Register to TR3-5	0F 24 [11 eee reg]		5	5		
TR3-5 to Register	0F 26 [11 eee reg]		10	10		
Register to TR6-TR7	0F 24 [11 eee reg]		6	6		
TR6-TR7 to Register						
MOVS *Move String*	A [010w]	- - - - - - - - - -	4	4	b	h
MOVSX *Move with Sign Extension*		- - - - - - - - - -			b	h
Register from Register/Memory	0F B[111w] [mod reg r/m]		1	1		
MOVZX *Move with Zero Extension*		- - - - - - - - - -			b	h
Register from Register/Memory	0F B[011w] [mod reg r/m]		1	1		
MUL *Unsigned Multiply*	F [011w] [mod 100 r/m]	x - - - x x u u x			b	h
Accumulator with Register/Memory						
Multiplier: Byte			4	4		
Word			4	4		
Doubleword			10	10		
NEG *Negate Integer*	F [011w] [mod 011 r/m]	x - - - x x x x x	1	1	b	h
NOP *No Operation*	90	- - - - - - - - - -	1	1		
NOT *Boolean Complement*	F [011w] [mod 010 r/m]	- - - - - - - - - -	1	1	b	h
OIO *Official Invalid OpCode*	0F FF	- - x 0 - - - - - -	1	8 - 125		
OR *Boolean OR*		0 - - - x x u x 0			b	h
Register to Register	0 [10dw] [11 reg r/m]		1	1		
Register to Memory	0 [100w] [mod reg r/m]		1	1		
Memory to Register	0 [101w] [mod reg r/m]		1	1		
Immediate to Register/Memory	8 [00sw] [mod 001 r/m] ###		1	1		
Immediate to Accumulator	0 [110w] ###		1	1		
OUT *Output to Port*		- - - - - - - - - -				m
Fixed Port	E [011w] #		14	14/28		
Variable Port	E [111w]		14	14/28		
OUTS *Output String*	6 [111w]	- - - - - - - - - -	14	14/28	b	h,m
POP *Pop Value off Stack*		- - - - - - - - - -			b	h,i,j
Register/Memory	8F [mod 000 r/m]		1	1		
Register (short form)	5 [1 reg]		1	1		
Segment Register (ES, SS, DS)	[000 sreg2 111]		1	3		
Segment Register (FS, GS)	0F [10 sreg3 001]		1	3		

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
\- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined
\+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

Table 6-21. Cyrix III 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
POPA *Pop All General Registers*	61	- - - - - - - - -	6	6	b	h
POPF *Pop Stack into FLAGS*	9D	x x x x x x x x x	9	9	b	h,n
PREFIX BYTES		- - - - - - - - -				m
Assert Hardware LOCK Prefix	F0					
Address Size Prefix	67					
Operand Size Prefix	66					
Segment Override Prefix						
CS	2E					
DS	3E					
ES	26					
FS	64					
GS	65					
SS	36					
PUSH *Push Value onto Stack*		- - - - - - - - -			b	h
Register/Memory	FF [mod 110 r/m]		1	1		
Register (short form)	5 [0 reg]		1	1		
Segment Register (ES, CS, SS, DS)	[000 sreg2 110]		1	1		
Segment Register (FS, GS)	0F [10 sreg3 000]		1	1		
Immediate	6 [10s0] ###		1	1		
PUSHA *Push All General Registers*	60	- - - - - - - - -	6	6	b	h
PUSHF *Push FLAGS Register*	9C	- - - - - - - - -	2	2	b	h
RCL *Rotate Through Carry Left*					b	h
Register/Memory by 1	D [000w] [mod 010 r/m]	x - - - - - - - - x	3	3		
Register/Memory by CL	D [001w] [mod 010 r/m]	u - - - - - - - - x	8	8		
Register/Memory by Immediate	C [000w] [mod 010 r/m] #	u - - - - - - - - x	8	8		
RCR *Rotate Through Carry Right*					b	h
Register/Memory by 1	D [000w] [mod 011 r/m]	x - - - - - - - - x	4	4		
Register/Memory by CL	D [001w] [mod 011 r/m]	u - - - - - - - - x	9	9		
Register/Memory by Immediate	C [000w] [mod 011 r/m] #	u - - - - - - - - x	9	9		
RDMSR *Read Model Specific Register*	0F 32	- - - - - - - - -				
RDPMC *Read Performance-Monitoring Counters*	0F 33	- - - - - - - - -				
RDSHR *Read SMM Header Pointer Register*	0F 36	- - - - - - - - -				
RDTSC *Read Time Stamp Counter*	0F 31	- - - - - - - - -				
REP INS *Input String*	F3 6[110w]	- - - - - - - - -	12+5n	12+5n\ 28+5n	b	h,m
REP LODS *Load String*	F3 A[110w]	- - - - - - - - -	10+n	10+n	b	h
REP MOVS *Move String*	F3 A[010w]	- - - - - - - - -	9+n	9+n	b	h

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined
\+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

Table 6-21. Cyrix III 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
REP OUTS *Output String*	F3 6[111w]	- - - - - - - - -	12+5n	12+5n\ 28+5n	b	h,m
REP STOS *Store String*	F3 A[101w]	- - - - - - - - -	10+n	10+n	b	h
REPE CMPS *Compare String (Find non-match)*	F3 A[011w]	x - - - x x x x x	10+2n	10+2n	b	h
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*		- - - - - - - - -			b	g,h,j,k,r
Within Segment	C3		3	3		
Within Segment Adding Immediate to SP			4	4		
Intersegment	C2 ##		4	7		
Intersegment Adding Immediate to SP	CB		4	7		
Protected Mode: Different Privilege Level	CA ##					
Intersegment				23		
Intersegment Adding Immediate to SP				23		
ROL *Rotate Left*					b	h
Register/Memory by 1	D[000w] [mod 000 r/m]	x - - - - - - - - x	1	1		
Register/Memory by CL	D[001w] [mod 000 r/m]	u - - - - - - - - x	2	2		
Register/Memory by Immediate	C[000w] [mod 000 r/m] #	u - - - - - - - - x	1	1		
ROR *Rotate Right*					b	h
Register/Memory by 1	D[000w] [mod 001 r/m]	x - - - - - - - - x	1	1		
Register/Memory by CL	D[001w] [mod 001 r/m]	u - - - - - - - - x	2	2		
Register/Memory by Immediate	C[000w] [mod 001 r/m] #	u - - - - - - - - x	1	1		
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*					b	h
Register/Memory by 1	D[000w] [mod 100 r/m]	x - - - x x u x x	1	1		
Register/Memory by CL	D[001w] [mod 100 r/m]	u - - - x x u x x	2	2		
Register/Memory by Immediate	C[000w] [mod 100 r/m] #	u - - - x x u x x	1	1		

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
\- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined
\+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

Table 6-21. Cyrix III 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
SAR *Shift Right Arithmetic*					b	h
Register/Memory by 1	D[000w] [mod 111 r/m]	x - - - x x u x x	1	1		
Register/Memory by CL	D[001w] [mod 111 r/m]	u - - - x x u x x	2	2		
Register/Memory by Immediate	C[000w] [mod 111 r/m] #	u - - - x x u x x	1	1		
SBB *Integer Subtract with Borrow*		x - - - x x x x x			b	h
Register to Register	1[10dw] [11 reg r/m]		1	1		
Register to Memory	1[100w] [mod reg r/m]		1	1		
Memory to Register	1[101w] [mod reg r/m]		1	1		
Immediate to Register/Memory	8[00sw] [mod 011 r/m] ###		1	1		
Immediate to Accumulator (short form)	1[110w] ###		1	1		
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

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined

\+ = 8-bit signed displacement

+++ = full signed displacement (16, 32 bits)

Table 6-21. Cyrix III 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
SETNLE/SETG *Set Byte on Not Less or Equal/Greater*		- - - - - - - - -				h
To Register/Memory	0F 9F [mod 000 r/m]		1	1		
SETNO *Set Byte on Not Overflow*		- - - - - - - - -				h
To Register/Memory	0F 91 [mod 000 r/m]		1	1		
SETNP/SETPO *Set Byte on Not Parity/Parity Odd*		- - - - - - - - -				h
To Register/Memory	0F 9B [mod 000 r/m]		1	1		
SETNS *Set Byte on Not Sign*		- - - - - - - - -				h
To Register/Memory	0F 99 [mod 000 r/m]		1	1		
SETO *Set Byte on Overflow*		- - - - - - - - -				h
To Register/Memory	0F 90 [mod 000 r/m]		1	1		
SETP/SETPE *Set Byte on Parity/Parity Even*		- - - - - - - - -				h
To Register/Memory	0F 9A [mod 000 r/m]		1	1		
SETS *Set Byte on Sign*		- - - - - - - - -				h
To Register/Memory	0F 98 [mod 000 r/m]		1	1		
SGDT *Store GDT Register*		- - - - - - - - -	4	4	b,c	h
To Register/Memory	0F 01 [mod 000 r/m]					
SHL *Shift Left Logical*					b	h
Register/Memory by 1	D [000w] [mod 100 r/m]	x - - - x x u x x	1	1		
Register/Memory by CL	D [001w] [mod 100 r/m]	u - - - x x u x x	2	2		
Register/Memory by Immediate	C [000w] [mod 100 r/m] #	u - - - x x u x x	1	1		
SHLD *Shift Left Double*		u - - - x x u x x			b	h
Register/Memory by Immediate	0F A4 [mod reg r/m] #		4	4		
Register/Memory by CL	0F A5 [mod reg r/m]		5	5		
SHR *Shift Right Logical*					b	h
Register/Memory by 1	D [000w] [mod 101 r/m]	x - - - x x u x x	1	1		
Register/Memory by CL	D [001w] [mod 101 r/m]	u - - - x x u x x	2	2		
Register/Memory by Immediate	C [000w] [mod 101 r/m] #	u - - - x x u x x	1	1		
SHRD *Shift Right Double*		u - - - x x u x x			b	h
Register/Memory by Immediate	0F AC [mod reg r/m] #		4	4		
Register/Memory by CL	0F AD [mod reg r/m]		5	5		
SIDT *Store IDT Register*		- - - - - - - - -			b,c	h
To Register/Memory	0F 01 [mod 001 r/m]		4	4		
SLDT *Store LDT Register*		- - - - - - - - -			a	h
To Register/Memory	0F 00 [mod 000 r/m]			1		
SMINT *Software SMM Entry*	0F 38	- - - - - - - - -	55	55	s	s

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined

\+ = 8-bit signed displacement

+++ = full signed displacement (16, 32 bits)

Table 6-21. Cyrix III 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
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
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

\# = immediate 8-bit data
x = modified
= immediate 16-bit data
- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined

\+ = 8-bit signed displacement

+++ = full signed displacement (16, 32 bits)

Table 6-21. Cyrix III 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
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
x = modified
= immediate 16-bit data
\- = unchanged
= full immediate 32-bit data (8, 16, 32 bits)
u = undefined
\+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

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 (CR0) 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 6-187). 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. Cyrix III FPU Instruction Set Summary

FPU INSTRUCTION	OP CODE	OPERATION	CLOCK COUNT	NOTES
F2XM1 *Function Evaluation 2^x-1*	D9 F0	TOS ⟵ 2^{TOS}-1	92 - 108	See
FABS *Floating Absolute Value*	D9 E1	TOS ⟵ | TOS |	2	Note 2
FADD *Floating Point Add*				
Top of Stack	DC [1100 0 n]	ST(n) ⟵ ST(n) + TOS	4 - 7	
80-bit Register	D8 [1100 0 n]	TOS ⟵ TOS + ST(n)	4 - 7	
64-bit Real	DC [mod 000 r/m]	TOS ⟵ TOS + M.DR	4 - 7	
32-bit Real	D8 [mod 000 r/m]	TOS ⟵ TOS + M.SR	4 - 7	
FADDP *Floating Point Add, Pop*	DE [1100 0 n]	ST(n) ⟵ ST(n) + TOS; then pop TOS	4 - 7	
FIADD *Floating Point Integer Add*				
32-bit integer	DA [mod 000 r/m]	TOS ⟵ TOS + M.SI	8 - 12	
16-bit integer	DE [mod 000 r/m]	TOS ⟵ TOS + M.WI	8 - 12	
FCHS *Floating Change Sign*	D9 E0	TOS ⟵ - 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]		9 - 10	
16-bit integer	DE [mod 010 r/m]	CC set by TOS - M.WI	9 - 10	
FICOMP *Floating Point Compare*		CC set by TOS - M.SI		
32-bit integer	DA [mod 011 r/m]		9 - 10	
16-bit integer	DE [mod 011 r/m]	CC set by TOS - M.WI; then pop TOS	9 - 10	
		CC set by TOS - M.SI; then pop TOS		
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*			9 - 10	
80-bit integer	DB [1110 1 n]	EFLAG set by TOS - ST(n)		
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) <—ST(n)	4	
FCMOVE *Floating Point Conditional Move if* Equal	DA [1100 1 n]	If (ZF=1) ST(0) <—ST(n)	4	

Table 6-23. Cyrix III 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	DC [1111 1 n]	ST(n) ⟵ ST(n) / TOS	24 - 34	
80-bit Register	D8 [1111 0 n]	TOS ⟵ TOS / ST(n)	24 - 34	
64-bit Real	DC [mod 110 r/m]	TOS ⟵ TOS / M.DR	24 - 34	
32-bit Real	D8 [mod 110 r/m]	TOS ⟵ TOS / M.SR	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	DC [1111 0 n]	TOS ⟵ ST(n) / TOS	24 - 34	
80-bit Register	D8 [1111 1 n]	ST(n) ⟵ TOS / ST(n)	24 - 34	
64-bit Real	DC [mod 111 r/m]	TOS ⟵ M.DR / TOS	24 - 34	
32-bit Real	D8 [mod 111 r/m]	TOS ⟵ M.SR / TOS	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	DA [mod 110 r/m]	TOS ⟵ TOS / M.SI	34 - 38	
16-bit Integer	DE [mod 110 r/m]	TOS ⟵ TOS / M.WI	33 - 38	
FIDIVR *Floating Point Integer Divide Reversed*				
32-bit Integer	DA [mod 111 r/m]	TOS ⟵ M.SI / TOS	34 - 38	
16-bit Integer	DE [mod 111 r/m]	TOS ⟵ M.WI / TOS	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	
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	

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

FPU INSTRUCTION	OP CODE	OPERATION	CLOCK COUNT	NOTES
FLD1 *Load Floating Const.= 1.0*	D9 E8	Push 1.0 onto stack	4	
FLDCW *Load FPU Mode Control Register* FLDENV *Load FPU Environment*	D9 [mod 101 r/m] D9 [mod 100 r/m]	Ctl ⟵ Memory Word Env Regs ⟵ Memory	4 30	
FLDL2E *Load Floating Const.= $Log_2(e)$* FLDL2T *Load Floating Const.= $Log_2(10)$* FLDLG2 *Load Floating Const.= $Log_{10}(2)$* FLDLN2 *Load Floating Const.= Ln(2)* FLDPI *Load Floating Const.= π* FLDZ *Load Floating Const.= 0.0*	D9 EA D9 E9 D9 EC D9 ED D9 EB D9 EE	Push $Log_2(e)$ onto stack Push $Log_2(10)$ onto stack Push $Log_{10}(2)$ onto stack Push $Log_e(2)$ onto stack Push π onto stack Push 0.0 onto stack	4 4 4 4 4 4	
FMUL *Floating Point Multiply* Top of Stack 80-bit Register 64-bit Real 32-bit Real FMULP *Floating Point Multiply & Pop* FIMUL *Floating Point Integer Multiply* 32-bit Integer 16-bit Integer	 DC [1100 1 n] D8 [1100 1 n] DC [mod 001 r/m] D8 [mod 001 r/m] DE [1100 1 n] DA [mod 001 r/m] DE [mod 001 r/m]	 ST(n) ⟵ ST(n) × TOS TOS ⟵ TOS × ST(n) TOS ⟵ TOS × M.DR TOS ⟵ TOS × M.SR ST(n) ⟵ ST(n) × TOS; then pop TOS TOS ⟵ TOS × M.SI TOS ⟵ TOS × M.WI	 4 - 6 4 - 6 4 - 6 4 - 5 4 - 6 9 - 11 8 - 10	
FNOP *No Operation*	D9 D0	No Operation	2	
FPATAN *Function Eval: $Tan^{-1}(y/x)$* FPREM *Floating Point Remainder* FPREM1 *Floating Point Remainder IEEE* FPTAN *Function Eval: Tan(x)* FRNDINT *Round to Integer*	D9 F3 D9 F8 D9 F5 D9 F2 D9 FC	ST(1) ⟵ ATAN[ST(1) / TOS]; then pop TOS TOS ⟵ Rem[TOS / ST(1)] TOS ⟵ Rem[TOS / ST(1)] TOS ⟵ TAN(TOS); then push 1.0 onto stack TOS ⟵ Round(TOS)	97 - 161 82 - 91 82 - 91 117 - 129 10 - 20	See Note 3 See Note 1
FRSTOR *Load FPU Environment and Reg.* FSAVE *Save FPU Environment and Reg* FNSAVE *Save FPU Environment and Reg*	DD [mod 100 r/m] (9B)DD [mod 110 r/m] DD [mod 110 r/m]	Restore state. Wait then save state. Save state.	56 - 72 57 - 67 55 - 65	
FSCALE *Floating Multiply by 2^n* FSIN *Function Evaluation: Sin(x)*	D9 FD D9 FE	TOS ⟵ TOS × $2^{(ST(1))}$ TOS ⟵ SIN(TOS)	7 - 14 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. Cyrix III 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.WI ⟵ TOS	8 - 13	
16-bit Integer	DF [mod 010 r/m]	M.SI ⟵ TOS	7 - 10	
FISTP *Store Integer FPU Register, Pop*				
64-bit Integer	DF [mod 111 r/m]	M.WI ⟵ TOS; then pop TOS	10 - 13	
32-bit Integer	DB [mod 011 r/m]	M.LI ⟵ TOS; then pop TOS	8 - 13	
16-bit Integer	DF [mod 011 r/m]	M.SI ⟵ TOS; then pop TOS	7 - 10	
FSTCW *Store FPU Mode Control Register*	(9B)D9 [mod 111 r/m]	Wait ⟵ Control Mode Register	5	
FNSTCW *Store FPU Mode Control Register*		Memory ⟵ Control Mode Register	3	
FSTENV *Store FPU Environment*	D9 [mod 111 r/m]	Memory ⟵ Env. Registers	14 - 24	
FNSTENV *Store FPU Environment*	(9B)D9 [mod 110 r/m]	Memory ⟵ Env. Registers	12 - 22	
FSTSW *Store FPU Status Register*		Memory ⟵ Status Register	6	
FNSTSW *Store FPU Status Register*	D9 [mod 110 r/m]	Wait ⟵ Status Register	4	
FSTSW AX *Store FPU Status Register to AX*	(9B)DD [mod 111 r/m]	Memory ⟵ Status Register	4	
FNSTSW AX *Store FPU Status Register to AX*	DD [mod 111 r/m]	Memory ⟵ Status Register	2	
	(9B)DF E0	Memory		
	DF E0	Wait		
		Memory		
		Memory		
		Wait		
		AX		
		AX		
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. Cyrix III FPU Instruction Set Summary (Continued)

FPU INSTRUCTION	OP CODE	OPERATION	CLOCK COUNT	NOTES
FSUBR *Floating Point Subtract Reverse*				
Top of Stack	DC [1110 0 n]	TOS ⟵ ST(n) - TOS	4 - 7	
80-bit Register	D8 [1110 1 n]	ST(n) ⟵ TOS - ST(n)	4 - 7	
64-bit Real	DC [mod 101 r/m]	TOS ⟵ M.DR - TOS	4 - 7	
32-bit Real	D8 [mod 101 r/m]	TOS ⟵ M.SR - TOS	4 - 7	
FSUBRP *Floating Point Subtract Reverse, Pop*	DE [1110 0 n]	ST(n) ⟵ TOS - ST(n); then pop TOS	4 - 7	
FISUB *Floating Point Integer Subtract*				
32-bit Integer	DA [mod 100 r/m]	TOS ⟵ TOS - M.SI	14 - 29	
16-bit Integer	DE [mod 100 r/m]	TOS ⟵ TOS - M.WI	14 - 27	
FISUBR *Floating Point Integer Subtract Reverse*				
32-bit Integer Reversed	DA [mod 101 r/m]	TOS ⟵ M.SI - TOS	14 - 29	
16-bit Integer Reversed	DE [mod 101 r/m]	TOS ⟵ M.WI - TOS	14 - 27	
FTST *Test Top of Stack*	D9 E4	CC set by TOS - 0.0	4	
FUCOM *Unordered Compare*	DD [1110 0 n]	CC set by TOS - ST(n)	4	
FUCOMP *Unordered Compare, Pop*	DD [1110 1 n]	CC set by TOS - ST(n); then pop TOS	4	
FUCOMPP *Unordered Compare, Pop two elements*	DA E9	CC set by TOS - ST(I); then pop TOS and ST(1)	4	
FWAIT *Wait*	9B	Wait for FPU not busy	2	
FXAM *Report Class of Operand*	D9 E5	CC ⟵ Class of TOS	4	
FXCH *Exchange Register with TOS*	D9 [1100 1 n]	TOS ⟷ ST(n) Exchange	2	
FXTRACT *Extract Exponent*	D9 F4	temp ⟵ TOS; TOS ⟵ exponent (temp); then push significant (temp) onto stack	11 - 16	
FLY2X *Function Eval. y × Log2(x)*	D9 F1	ST(1) ⟵ ST(1) × Log_2(TOS); then pop TOS	145 - 154	
FLY2XP1 *Function Eval. y × Log2(x+1)*	D9 F9	ST(1) ⟵ ST(1) × Log_2(1+TOS); then pop TOS	131 - 133	See Note 4

FPU Instruction Clock Counts

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 < 3π/4.
Add 90 clock counts for argument reduction if outside this range.

For FCOS, clock count is 141 if TOS < π/4 and clock count is 92 if π/4 < TOS > π/2.

For FSIN, clock count is 81 to 82 if absolute value of TOS < π/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 < π/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 Cyrix III Processor MMX Instruction Clock Counts

The CPU is functionally divided into the FPU unit, and the integer unit. The FPU has been extended to processes 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 continues 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 6-194). The abbreviations used in this table are listed in Table 6-22.

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. Cyrix III Processor MMX Instruction Set Clock Count Summary

MMX INSTRUCTIONS	OPCODE	OPERATION	CLOCK COUNT LATENCY /THROUGHPUT
EMMS *Empty MMX State*	0F77	Tag Word <--- FFFFh (empties the floating point tag word)	1/1
MOVD *Move Doubleword*			
Register to MMX Register	0F6E [11 mm reg]	MMX reg [qword] <--move, zero extend-- reg [dword]	1/1
MMX Register to Register	0F7E [11 mm reg]	reg [qword] <--move-- MMX reg [low dword]	5/1
Memory to MMX Register	0F6E [mod mm r/m]	MMX regr[qword] <--move, zero extend-- memory[dword]	1/1
MMX Register to Memory	0F7E [mod mm r/m]	Memory [dword] <--move-- MMX reg [low dword]	1/1
MOVQ *Move Quardword*			
MMX Register 2 to MMX Register 1	0F6F [11 mm1 mm2]	MMX reg 1 [qword] <--move-- MMX reg 2 [qword]	1/1
MMX Register 1 to MMX Register 2	0F7F [11 mm1 mm2]	MMX reg 2 [qword] <--move-- MMX reg 1 [qword]	1/1
Memory to MMX Register	0F6F [mod mm r/m]	MMX reg [qword] <--move-- memory[qword]	1/1
MMX Register to Memory	0F7F [mod mm r/m]	Memory [qword] <--move-- MMX reg [qword]	1/1
PACKSSDW *Pack Dword with Signed Saturation*	0F6B [11 mm1 mm2]	MMX reg 1 [qword] <--packdw, signed sat-- MMX reg 2, MMX reg 1	1/1
MMX Register 2 to MMX Register 1	0F6B [mod mm r/m]	MMX reg [qword] <--packdw, signed sat-- memory, MMX reg	1/1
Memory to MMX Register			
PACKSSWB *Pack Word with Signed Saturation*	0F63 [11 mm1 mm2]	MMX reg 1 [qword] <--packwb, signed sat-- MMX reg 2, MMX reg 1	1/1
MMX Register 2 to MMX Register 1	0F63 [mod mm r/m]	MMX reg [qword] <--packwb, signed sat-- memory, MMX reg	1/1
Memory to MMX Register			
PACKUSWB *Pack Word with Unsigned Saturation*	0F67 [11 mm1 mm2]	MMX reg 1 [qword] <--packwb, unsigned sat-- MMX reg 2, MMX reg 1	1/1
MMX Register 2 to MMX Register 1	0F67 [mod mm r/m]	MMX reg [qword] <--packwb, unsigned sat-- memory, MMX reg	1/1
Memory to MMX Register			
PADDB *Packed Add Byte with Wrap-Around*			
MMX Register 2 to MMX Register 1	0FFC [11 mm1 mm2]	MMX reg 1 [byte] <---- MMX reg 1 [byte] + MMX reg 2 [byte]	1/1
Memory to MMX Register	0FFC [mod mm r/m]	MMX reg[byte] <---- memory [byte] + MMX reg [byte]	1/1
PADDD *Packed Add Dword with Wrap-Around*	0FFE [11 mm1 mm2]	MMX reg 1 [sign dword] <---- MMX reg 1 [sign dword] + MMX reg 2 [sign dword]	1/1
MMX Register 2 to MMX Register 1	0FFE [mod mm r/m]	MMX reg [sign dword] <---- memory [sign dword] + MMX reg [sign dword]	1/1
Memory to MMX Register			
PADDSB *Packed Add Signed Byte with Saturation*	0FEC [11 mm1 mm2]	MMX reg 1 [sign byte] <--sat-- MMX reg 1 [sign byte] + MMX reg 2 [sign byte]	1/1
MMX Register 2 to MMX Register1	0FEC [mod mm r/m]	MMX reg [sign byte] <--sat-- memory [sign byte] + MMX reg [sign byte]	1/1
Memory to Register			
PADDSW *Packed Add Signed Word with Saturation*	0FED [11 mm1 mm2]	MMX reg 1 [sign word] <--sat-- MMX reg 1 [sign word] + MMX reg 2 [sign word]	1/1
MMX Register 2 to MMX Register1	0FED [mod mm r/m]	MMX reg [sign word] <--sat-- memory [sign word] + MMX reg [sign word]	1/1
Memory to Register			
PADDUSB *Add Unsigned Byte with Saturation*	0FDC [11 mm1 mm2]	MMX reg 1 [byte] <--sat-- MMX reg 1 [byte] + MMX reg 2 [byte]	1/1
MMX Register 2 to MMX Register1	0FDC [mod mm r/m]	MMX reg [byte] <--sat-- memory [byte] + MMX reg [byte]	1/1
Memory to Register			

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

MMX INSTRUCTIONS	OPCODE	OPERATION	CLOCK COUNT LATENCY /THROUGHPUT
PADDUSW *Add Unsigned Word with Saturation* MMX Register 2 to MMX Register1 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
PADDW *Packed Add Word with Wrap-Around* 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 *Bitwise Logical AND* 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 *Bitwise Logical AND NOT* 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 *Packed Byte Compare for Equality* 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 *Packed Dword Compare for Equality* 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 *Packed Word Compare for Equality* 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 *Pack Compare Greater Than Byte* 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

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

MMX INSTRUCTIONS	OPCODE	OPERATION	CLOCK COUNT LATENCY /THROUGHPUT
PCMPGTD *Pack Compare Greater Than Dword* 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
PCMPGTW *Pack Compare Greater Than Word* MMX Register 2 to MMX Register1 Memory with MMX Register	0F65 [11 mm1 mm2] 0F65 [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
PMADDWD *Packed Multiply and Add* 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 *Packed Multiply High* 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 *Packed Multiply Low* 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 *Bitwise OR* 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 *Packed Shift Left Logical Dword* 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 *Packed Shift Left Logical Qword* 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 *Packed Shift Left Logical Word* 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 *Packed Shift Right Arithmetic* Dword 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
PSRAW *Packed Shift Right Arithmetic* Word 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

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

MMX INSTRUCTIONS	OPCODE	OPERATION	CLOCK COUNT LATENCY /THROUGHPUT
PSRLD *Packed Shift Right Logical Dword*			
MMX Register 1 by MMX Register 2	0FD2 [11 mm1 mm2]	MMX reg 1 [dword] <--shift right, shifting in zeroes by MMX reg 2 [dword]--	1/1
MMX Register by Memory	0FD2 [mod mm r/m]	MMX reg [dword] <--shift right, shifting in zeroes by memory[dword]--	1/1
MMX Register by Immediate	0F72 [11 010 mm] #	MMX reg [dword] <--shift right, shifting in zeroes by [im byte]--	1/1
PSRLQ *Packed Shift Right Logical Qword*			
MMX Register 1 by MMX Register 2	0FD3 [11 mm1 mm2]	MMX reg 1 [qword] <--shift right, shifting in zeroes by MMX reg 2 [qword]	1/1
MMX Register by Memory	0FD3 [mod mm r/m]	MMX reg [qword] <--shift right, shifting in zeroes by memory[qword]	1/1
MMX Register by Immediate	0F73 [11 010 mm] #	MMX reg [qword] <--shift right, shifting in zeroes by [im byte]	1/1
PSRLW *Packed Shift Right Logical Word*			
MMX Register 1 by MMX Register 2	0FD1 [11 mm1 mm2]	MMX reg 1 [word] <--shift right, shifting in zeroes by MMX reg 2 [word]	1/1
MMX Register by Memory	0FD1 [mod mm r/m]	MMX reg [word] <--shift right, shifting in zeroes by memory[word]	1/1
MMX Register by Immediate	0F71 [11 010 mm] #	MMX reg [word] <--shift right, shifting in zeroes by imm[word]	1/1
PSUBB *Subtract Byte With Wrap-Around*			
MMX Register 2 to MMX Register1	0FF8 [11 mm1 mm2]	MMX reg 1 [byte] <---- MMX reg 1 [byte] subtract MMX reg 2 [byte]	1/1
Memory to MMX Register	0FF8 [mod mm r/m]	MMX reg [byte] <---- MMX reg [byte] subtract memory [byte]	1/1
PSUBD *Subtract Dword With Wrap-Around*			
MMX Register 2 to MMX Register1	0FFA [11 mm1 mm2]	MMX reg 1 [dword] <---- MMX reg 1 [dword] subtract MMX reg 2 [dword]	1/1
Memory to MMX Register	0FFA [mod mm r/m]	MMX reg [dword] <---- MMX reg [dword] subtract memory [dword]	1/1
PSUBSB *Subtract Byte Signed With Saturation*			
MMX Register 2 to MMX Register1	0FE8 [11 mm1 mm2]	MMX reg 1 [sign byte] <--sat-- MMX reg 1 [sign byte] subtract MMX reg 2 [sign byte]	1/1
Memory to MMX Register	0FE8 [mod mm r/m]	MMX reg [sign byte] <--sat-- MMX reg [sign byte] subtract memory [sign byte]	1/1
PSUBSW *Subtract Word Signed With Saturation*	0FE9 [11 mm1 mm2]	MMX reg 1 [sign word] <--sat--MMX reg 1 [sign word] subtract MMX reg 2 [sign word]	1/1
MMX Register 2 to MMX Register1	0FE9 [mod mm r/m]	MMX reg [sign word] <--sat-- MMX reg [sign word] subtract memory [sign word]	1/1
Memory to MMX Register			
PSUBUSB *Subtract Byte Unsigned With Saturation*	0FD8 [11 mm1 mm2]	MMX reg 1 [byte] <--sat-- MMX reg 1 [byte] subtract MMX reg 2 [byte]	1/1
MMX Register 2 to MMX Register1	0FD8 [11 mm reg]	MMX reg [byte] <--sat-- MMX reg [byte] subtract memory [byte]	1/1
Memory to MMX Register			
PSUBUSW *Subtract Word Unsigned With Saturation*	0FD9 [11 mm1 mm2]	MMX reg 1 [word] <--sat-- MMX reg 1 [word] subtract MMX reg 2 [word]	1/1
MMX Register 2 to MMX Register1	0FD9 [11 mm reg]	MMX reg [word] <--sat-- MMX reg [word] subtract memory [word]	1/1
Memory to MMX Register			
PSUBW *Subtract Word With Wrap-Around*			
MMX Register 2 to MMX Register1	0FF9 [11 mm1 mm2]	MMX reg 1 [word] <---- MMX reg 1 [word] subtract MMX reg 2 [word]	1/1
Memory to MMX Register	0FF9 [mod mm r/m]	MMX reg [word] <---- MMX reg [word] subtract memory [word]	1/1
PUNPCKHBW *Unpack High Packed Byte Data to Packed Words*			
MMX Register 2 to MMX Register1	0F68 [11 mm1 mm2]	MMX reg 1 [byte] <--interleave-- MMX reg 1 [up byte], MMX reg 2 [up byte]	1/1
Memory to MMX Register	0F68 [11 mm reg]	MMX reg [byte] <--interleave-- memory [up byte], MMX reg [up byte]	1/1
PUNPCKHDQ *Unpack High Packed Dword Data to Qword*			
MMX Register 2 to MMX Register1	0F6A [11 mm1 mm2]	MMX reg 1 [dword] <--interleave-- MMX reg 1 [up dword], MMX reg 2 [up dword]	1/1
Memory to MMX Register	0F6A [11 mm reg]	MMX reg [dword] <--interleave-- memory [up dword], MMX reg [up dword]	1/1

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

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

6.7 Cyrix III Processor 3DNow! Clock Counts

The CPU is functionally divided into the FPU unit, and the integer unit. The FPU has been extended to processes both MMX, and 3DNow! instructions in addition to floating point instructions in parallel with the integer unit.

For example, when the integer unit detects a 3DNow! instruction, the instruction passes to the FPU unit for execution. The integer unit continues to execute instructions while the FPU unit executes the 3DNow! instruction.

6.7.1 3DNow! Clock Count Table

The clock counts for the MMX instructions are listed in Table 6-25. (Page 6-194). The abbreviations used in this table are listed in Table 6-22.

Table 6-24. 3DNow! Clock Count Table Abbreviations

Operation	Function	Opcode Suffix	Delay	Thoughput
PF2ID	Packed FP to 32-bit Integer	1Dh	3	1
PFACC	Packed FP Accumulate	AEh	3	1
PFCMPEQ	Packed FP Comparison, Equal	B0h	3	1
PFCMPGE	Packed FP Comparison, Greater or Equal	90h	3	1
PFCMPGT	Packed FP Comparison, Greater	A0h	3	1
PFMAX	Packed FP Maximum	A4h	2	1
PFMIN	Packed FP Minimum	94h	2	1
PFMUL	Packed FP Multiplication	B4h	3	1
PFRCP	Packed FP Reciprocal Approximation	96h	5	3
PFRSQRT	Packed FP Reciprocal Square Root Approximation	97h	5	3
PFSUB	Packed FP Subtraction	9Ah	3	1
PFSUBR	Packed FP Reverse Subtraction	AAh	3	1
PI2FD	Packed 32-bit Integer to FP Conversion	0Dh	3	1
FEMMS	Empty MX/3DNow! State	0Eh	1	1
PAVGUSB	Packed 8-bit Unsigned Integer Averaging	BFh	1	1
PFADD	Packed FP Addition	9Eh	3	1
PFRCPIT1	Packed FP Reciprocal First Iteration Step	A6h	1	1
PFRSQIT1	Packed FP Reciprocal Square Root First Iteration Step	A7h	1	1
PFRCPIT2	Packed FP Reciprocal/Reciprocal Square Root Second Iteration Step	B6h	1	1
PMULHRW	Packed 16-bit Integer Multiply with rounding	B7h	2	1
PREFETCH	Not required, functions as NOP	0Dh	1	1

Cyrix III Processor 3DNow! Clock Counts