Application Note 120
Cyrix III CPU BIOS
Writer's Guide

Cyrix Processors

REVISION HISTORY

Date	Version	Revision
11/24/99	1.1	Updated register 49h and DIR1 table
5/20/99	1.0	Updated for Cyrix III
4/19/99	0.43	Format changes, chapter 6.
3/22/99	0.42	Changed processor internal code name from MXs to Cyrix III.
3/17/99	0.41	Page 55: Added Core to Bus Clock Ratio Configuration Register.
2/23/99	0.4	Added Confidential Notice. Made all pages same width.
1/29/99	0.3	Minor changes: Changed fonts, added revision page, changed DIR table titles
12/14/98	0.2	Cyrix III information added. Completed Table page 32
9/21/98	0.1	Intial First Draft based on App Notes 112 and 118. C:\documentation\joshua\appnotes\j120ap.fm (confidential)

Table of Contents

APPLICATION NOTE 120

Cyrix III BIOS Writer's Guide

1. Introduction

1.1 Scope

This document is intended for Cyrix III system BIOS writers. It is not a stand-alone document, but a supplement to other Cyrix documentation including the Cyrix III Data Books, and Cyrix SMM Programmer's Guide. Recommendations for Cyrix III detection and configuration register settings are included.

The recommended settings are optimized for performance and compatibility in Windows95 or Windows NT, Plug and Play (Pnp), PCI-based system. Performance optimization, CPU detection, chipset initialization, memory discovery, I/O recovery time, and other functions are described in detail.

1.2 Cyrix Configuration Registers

The Cyrix III uses on-chip configuration registers to control the on-chip cache, system management mode (SMM), device identification, and other Cyrix III specific features. The on-chip registers are used to activate advanced performance features. These performance features may be enabled “globally” in some cases, or by a user-defined address region. The flexible configuration of the Cyrix III is intended to fit a wide variety of systems.

BIOS needs to perform 3 basic functions outlined in this document. They are:

1) Identification of Cyrix III cpu, frequency, and performance rating.

2) Set up of Configuration Registers in the cpu to enable features, turn on cache, and set clock multiplier.

3) Set up Address Region Registers and Region Control Registers to control memory accesses.

The Importance of Non-Cacheable Regions

The Cyrix III has fourteen internal user-defined Address Region Registers and Region Control Registers. Among other attributes, the regions define cacheability of the address regions. Using this cacheability information, the Cyrix III is able to implement high performance features, that would otherwise not be possible. A non-cacheable region implies that read sourcing from the write buffers, data forwarding, data bypassing, speculative reads, and fill buffer streaming are disabled for memory accesses within that region. Additionally, strong cycle ordering is also enforced.

The Cyrix III also uses these Address Region Registers to setup the write gathering, or write combining, feature normally used for improving performance of video buffer memory. This feature is enabled differently from the Intel Celeron. Celeron uses machine specific registers (MSR’s) called memory type and range registers (MTRR’s). Cyrix III uses the Address Region Registers (ARR’s) and Region Control Registers (RCR’s) to perform this same feature. Thus BIOS, operating system calls, and video drivers should be updated with ARR and RCR setup instead of MTRR setup whenever Cyrix III is detected to enable write combining feature.

2. CPU Detection

The Cyrix III cpu can be identified using the CPU_ID instruction as explained below. It can also be identified using Cyrix specific Device Identification Registers (DIR). The methods for identifying the cpu and its available features are explained below. Once the Cyrix III cpu is identified, it should be named correctly as explained in the next section.

2.1 CPU Name and Performance Rating

The Cyrix III uses the performance rating system of speed measurement and reporting. The following table is used to identify the performance rating of the Cyrix III compared to actual Mhz. The performance rating is achieved by benchmarking the Cyrix III vs. a Celeron cpu in the same configuration.

CPU Name and PR rating	Bus Speed Mhz	Core Speed Mhz	Clock Multiplier
Cyrix III - 433	100	350	3.5
Cyrix III - 466	122	366	3.0
Cyrix III - 500	133	400	3.0
Cyrix III - 533	124	433	3.5
Cyrix III - 533	100	450	4.5
Cyrix III - 566	133	466	3.5
Cyrix III - 600	100	500	5.0

2.2 Cyrix CPU Identification and Inquiry Flow Chart

The Cyrix CPU Identification process (Figure1 on page6) consists of up to three tests and three inquiries. If CPUID is not supported, a 5/2 Test is performed to check if the CPU is a Cyrix part. If CPUID is supported, a second test is made to see if extended level CPUID is supported.

The numbers in parenthesis shown in Figure 1 refer to the supporting paragraphs in this manual.

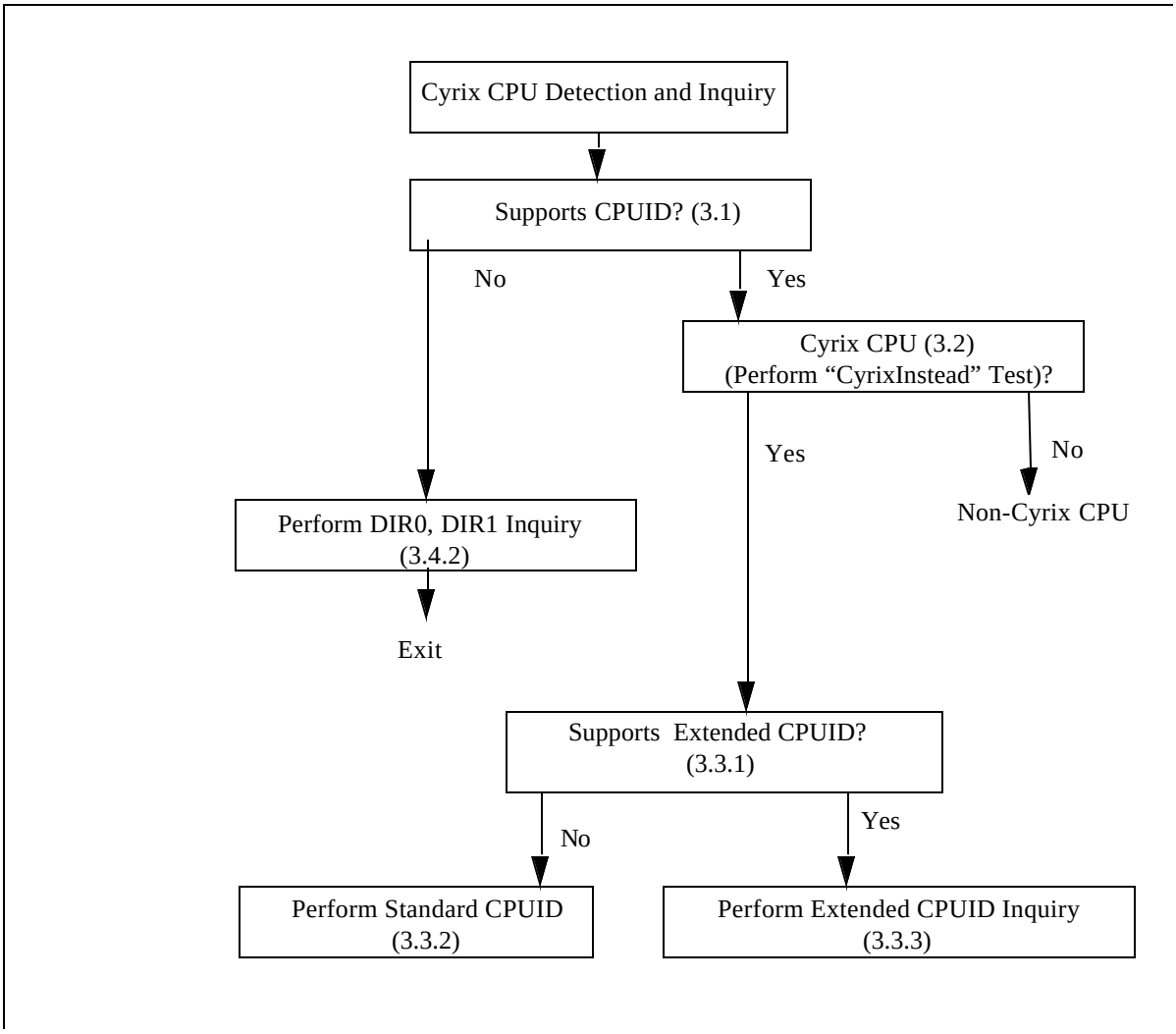


FIGURE 1. CPU Identification and Inquiry

Note: **The testing must be performed in the order shown in Figure 1 or testing may be invalid**.

2.3 CPU Detection Steps: CPUID Support Test

In order to avoid an invalid opcode exception on processors that do not support the CPUID instruction, software must first verify that the processor supports the CPUID instruction. The presence of the CPUID instruction is indicated by the ID bit (bit 21) in the EFLAGS register. If this bit can be toggled, the CPUID instruction is present and enabled on the processor. The following code will check for the presence of the CPUID instruction.
CPUID Support Test Sample Code*:

```
pushfd              ; get extended flags
pop     eax         ; store extended flags in eax
mov     ebx, eax    ; save current flags
xor     eax, 200000h    ; toggle bit 21
push    eax         ; put new flags on stack
popfd               ; flags updated now in flags
pushfd              ; get extended flags
pop     eax         ; store extended flags in eax
xor     eax, ebx    ; if bit 21 r/w then eax <> 0
je      no_cpuid    ; can't toggle id bit (21) no cpuid here
```

*Note: It has been assumed that the tests for EFLAGS support has been complete prior to this point. If CPUID is supported, it can be assumed that the CPU is an 80486 or above class processor.

2.3.1 "CyrixInstead" Test

The CPUID instruction level 0 provides vendor information. Following execution of the CPUID instruction with an input value of "0" in EAX, the EBX, ECX and EDX registers contain the vendor string of the CPU. To verify that the processor is a Cyrix CPU, the software checks for "CyrixInstead" in the return registers as shown in the sample code below:

"CyrixInstead" Test Sample Code

```
mov     eax, 0                          ; CPUID standard level 0
cpuid
cmp     ebx, 'iryC'
jne     not_cyrix
cmp     edx, 'snIx'
jne     not_cyrix
cmp     ecx, 'daet'
jne     not_cyri
```

2.4 Standard and Extended CPUID Levels

The CPUID instruction has been extended on recent processors so that additional information can be obtained from the CPU concerning items including stepping, model, family, type, TLB and cache information. The original levels of the CPUID instruction are termed “the standard CPUID levels” and the newer levels are termed the “extended CPUID levels.”

The standard and extended CPUID levels differ in that the EAX register’s most significant bit is set for the extended CPUID levels. Both the standard and extended CPUID levels may be executed at any privilege level. The EAX register provides the input value for the CPUID instruction to indicate what information should be returned by the instruction.

2.4.1 Extended CPUID Level Support Testing

This test is performed to determine if the CPU supports the extended CPUID levels.

Extended CPUID Instruction support testing consists of executing a CPUID instruction with the EAX register initialized to 8000 0000h and testing the return value in EAX. If a value greater than or equal to 8000 0000h is returned to the EAX register by the CPUID instruction, the CPU can execute extended CPUID instructions. The following sample code tests for Extended CPUID support.

Extended CPUID Instruction Test Sample Code:

```
mov     eax, 80000000h    ; try extended cpuid level
cpuid                     ; execute cpuid instruction
cmp     eax, 80000000h  ; check if extended levels are supported
jb      no_extended     ; extended cpuid functions not available
```

2.4.2 Standard CPUID Instruction Inquiry

The CPUID instruction provides processor and feature set information. This instruction may be executed at any privilege level. The standard CPUID instruction is defined as a CPUID instruction with the EAX register initialized to one of the following values:

0000 0000h - maximum standard levels supported and vendor string
0000 0001h - family, model and stepping information
0000 0002h - cache and TLB information

Table 1. summarizes the CPUID values returned by standard CPUID levels on Cyrix III processors.

Table 1. Summary of Returned Standard CPUID Values

DESCRIPTION	INITIAL EAX VALUE	CYRIX III
Maximum Standard Value	0h	2h
Stepping	1h	xxh
Model	1h	5h
Family	1h	6h
Type	1h	0h
TLB/Cache	2h	xxh

Note: xx = stepping revision specific.

2.4.3 CPUID Instruction with EAX = 0000 0000h

Standard function 0h (EAX = 0) of the CPUID instruction returns the maximum standard CPUID levels supported by the current processor to the EAX register. The maximum standard CPUID level is the highest acceptable value for the EAX register input.
After the instruction is executed registers EBX through EDX contain the vendor string of the processor. Note that the middle section is placed (out of order) into the EDX register (Table 2).

Table 2. Standard CPUID with EAX = 0000 0000h

REGISTER*	CONTENTS
EAX	Max Standard Levels
EBX	Vendor ID String 1="CYRI"
EDX	Vendor ID String 2="XINS"
ECX	Vendor ID String 3="TEAD"

*Note: The register order is correct.

2.4.4 CPUID Instruction with EAX = 0000 0001h

Standard function 01h (EAX = 1) of the CPUID instruction returns the Processor Type, Family, Model, and Stepping information of the current processor in EAX (Table 3). The Standard Feature Flags supported are returned in the EDX register. The other registers upon return are currently reserved.

Table 3. Standard CPUID with EAX = 0000 0001h

REGISTER	CONTENTS
EAX[3:0]	Stepping ID=revision specific step id
EAX[7:4]	Model=5 h
EAX[11:8]	Family=6 h
EAX[15:12]	Type=0 h
EAX[31:16]	Reserved
EBX	Reserved
ECX	Reserved
EDX	Standard Feature Flags=0080A13D h

Standard Feature Flags

The standard feature flags are returned in the EDX register when the CPUID instruction is called with standard function 01h (EAX = 1). Each flag refers to a specific feature and indicates if that feature is present on the processor. Some of these features require enabling or have protection control in CR4. Table 4. summarizes the standard feature flags.

Before using any of these features on the processor, the software should check the corresponding feature flag (Table 4). Attempting to execute an unavailable feature can cause exceptions and unexpected behavior. For example, software must check bit 4 before attempting to use the Time Stamp Counter instruction.

Table 4. Standard Feature Flags Values Returned in EDX

FEATURE FLAG	EDX BIT	CR4 BIT	CYRIX III
FPU On-Chip	0	-	1
Virtual Mode Extensions (V86)	1	0,1	01
Debug Extension	2	3	1
4 MB Page Size	3	4	1
Time Stamp Counter	4	2	1
RDMSR/WRMSR Instructions	5	8	1
Physical Address Extensions	6	5	0
Machine Check Exception	7	6	0
CMPXCHG8B Instruction Support	8	-	1
On-chip APIC Hardware	9	-	0
Reserved	10	-	0
SYSENTER/SYSEXIT Instructions	11	-	0
Memory Type Range Registers (MTRR)	12	-	0
Page Global Enable (PTE-PGE)	13	7	1
Machine Check Architecture	14	-	0
Conditional Move Instruction (CMOV)	15	-	1
Page Attribute Table	16	-	0
36-Bit Page Size Extensions	17	-	0
Reserved	18-22	-	00000
MMX™ Instructions	23	-	1
Fast FPU Save and Restore	24	-	0
Reserved	25-31	-	0000000

*Note: The CPUID instruction is disabled by default.

2.4.5 CPUID Instruction with EAX = 0000 0002h

Standard function 02h (EAX = 02h) of the CPUID instruction returns information that is specific to the Cyrix family of processors. Information about the TLB is returned in EAX. Information about the L1 Cache is returned in EDX. This information is to be looked up in a lookup table. See Table 13 on page 19.

Table 5. Standard CPUID with EAX = 0000 0002h

REGISTER	CONTENTS
EAX	TLB Information = 00747701 h
EBX	Reserved
ECX	Reserved
EDX	L1 Cache Information = 00008242 h

2.4.6 Extended CPUID Levels

The extended CPUID Instruction is defined when the EAX register is initialized to one of the following values (Table 6.):

8000 0000h - Maximum Levels
8000 0001h - Processor Information/Extended features
8000 0002h - Processor Marketing Name
8000 0003h - Processor Marketing Name
8000 0004h - Processor Marketing Name
8000 0005h - TLB/Cache Information

Each of the extended CPUID levels reports information that is specific to the Cyrix family of processors.

Table 6. Summary of Returned Extended CPUID Values

DESCRIPTION	INITIAL EAX VALUE	CYRIX III
Extended Levels	8000 0000h	5 h
TLB Info	8000 0005h	TBD
Cache Info	8000 0005h	TBD

*Note: The CPUID instruction is disabled by default.

Table 7. Summary of CPUID Functions

EXTENDED FUNCTION	DESCRIPTION	CYRIX III
8000 0000h	Extended Levels	X
8000 0001h	Extended Processor Info. Extended Feature Flags	X
8000 0002h	Processor Marketing Name	X
8000 0003h	Processor Marketing Name	X
8000 0004h	Processor Marketing Name	X
8000 0005h	TLB & Cache Information	X

2.4.7 CPUID Instruction with EAX = 8000 0000h

Extended function 8000 0000h (EAX = 8000 0000h) of the CPUID instruction returns the maximum extended CPUID levels supported by the current processor in EAX (Table 8). The other registers are currently reserved.

Table 8. Maximum Extended CPUID Level

REGISTER	CONTENTS
EAX	Maximum Extended Levels = 5 h
EBX	Reserved
ECX	Reserved
EDX	Reserved

2.4.8 CPUID Instruction with EAX = 8000 0001h

Extended function 8000 0001h (EAX = 8000 0001h) of the CPUID instruction returns the Processor Type, Family, Model, and Stepping information of the current processor in EAX (Table 9). The Extended Feature Flags supported are returned in EDX. The other registers are currently reserved.

Table 9. . Processor Signature and Extended Feature Flags

REGISTER	CONTENTS
EAX[3:0]	Stepping ID = revision specific step id
EAX[7:4]	Model = 5
EAX[11:8]	Family = 6
EAX[15:12]	Processor Type = 0
EAX[31:16]	Reserved
EBX	Reserved
ECX	Reserved
EDX	Extended Feature Flags = 0080A13D

Extended Feature Flags

The extended feature flags are returned in the EDX register when the CPUID instruction is called with extended function 8000 0001h (EAX = 8000 0001h). Each flag refers to a specific feature and indicates if that feature is present on the processor. Some of these features require enabling or have protection control in CR4. Table 10. summarizes the extended feature flags.

Table 10. Extended Feature Flags

FEATURE FLAG	EDX BIT	CR4 BIT	CYRIX III
Floating Point Unit	0	-	1
Virtual Mode Extensions (V86)	1	0,1	01
Debug Extension	2	3	-
Page Size Extensions (4 MByte)	3	4	-
Time Stamp Counter	4	2	X
Cyrix Model-Specific Registers (MSR)	5	8	X
Reserved	6	-	-
Machine Check Exception	7	6	-
CMPXCHG8B Instruction	8	-	X
SYSCALL and SYSRET Instructions	11	-	-
Reserved	12	-	-
Global Paging Extension (PTE-PGE)	13	7	-
Reserved	14	-	-
Integer Conditional Move Instructions (CMOV)	15	-	X
Floating-Point Conditonal Move Instructions	16		
Reserved	17-22	-	-
MMX™ Instructions	23	-	X
Cyrix 6x86MX Multimedia Extensions	24	-	X
Reserved	25 - 30	-	-
3DNow!™ Instructions	31	-	X

2.4.9 CPUID Instruction with EAX = 8000 0002h - 8000 0004h

Extended functions 8000 0002h through 8000 0004h (EAX = 8000 0002h through EAX = 8000 0004h) of the CPUID instruction returns an ASCII string containing the name of the current processor (Table 11). These functions eliminate the need to look up the processor name in a lookup table. Software can simply call these functions to obtain the name of the processor. The string may be 48 ASCII characters long, and is returned in little endian format. If the name is shorter than 48 characters long, the remaining bytes will be filled with ASCII NUL character (00h).

Table 11. Official CPU Name

8000 0002H		8000 0003H		8000 0004H	
EAX	CPU Name 1	EAX	CPU Name 5	EAX	CPU Name 9
EBX	CYRI	EBX	CPU Name 6	EBX	CPU Name 10
ECX	X II	ECX	CPU Name 7	ECX	CPU Name 11
EDX	I(tm)	EDX	CPU Name 8	EDX	CPU Name 12

2.4.10 CPUID Instruction with EAX = 8000 0005h

Extended function 8000 0005h (EAX = 8000 0005h) of the CPUID instruction returns information about the TLB and L1 Cache to be looked up in a lookup table. Refer to Tables Figure 12 and Figure 13 shown below.

Table 12. Cache and TLB Information

REGISTER	CONTENTS
EAX	Reserved
EBX	TLB Information
ECX	L1 Cache Information
EDX	Reserved

Table 13. Cache and Descriptor Lookup Table

CPUID LEVEL	REGISTER	VALUE	COMMENTS
Standard	EAX	xx xx xx 01h	The CPUID instruction needs to be executed only once with an input value of 02h to retrieve complete information about the cache and TLB.
Extended	EBX		
Standard	EAX	xx xx 70 xxh	TLB is 32 Entry, 4-way set associative, and has 4 KByte Pages
Extended	EBX		
Standard	EDX	xx xx xx 80h	L1 cache is 16 KBytes, 4-way set associative, and has 16 bytes per line.
Extended	ECX		

2.5 Non-CPUID Testing and Inquiry

The Cyrix III processor supports CPUID to determine processor type. If it is a Cyrix processor, inquires may also be made to the Device Identification Registers (DIR0 and DIR1) to determine which Cyrix processor is present.

2.5.1 DIR0, DIR1 Inquiry

After determining that a Cyrix processor without CPUID exists, its Device ID Registers (DIR) can be read to identify the Cyrix processor type. The Device ID Registers are located using register indexes FEh and FFh. Access to these registers is achieved by writing the index of the register to I/O port 22h. I/O port 23h is then used for data transfer. Each port 23h data transfer must be preceded by a port 22h-register index selection; otherwise the second and later port 23h operations are directed off-chip and produce external I/O cycles. The Tables 14 and 15 describe the bit definitions for the DIR0 and DIR1 Registers.

Table 14. DIR0 Bit Definitions

BIT POSITION	DESCRIPTION
7 - 4	CPU Device Identification Number (read only)
3 - 0	CPU Clock Multiplier (read only)

Table 15. DIR1 Bit Definitions

BIT POSITION	DESCRIPTION
7 - 4	CPU Step Identification Number (read only)
3 - 0	CPU Revision Identification (read only)

Table 16. describes the range of DIR0 values for the different generations of Cyrix CPU's

Table 16. CPU Generation Values in DIR0

DIR0 VALUES	DESCRIPTION
80h - 8Fh	Cyrix III

Table 17. Cyrix III CPU DIR Values

DIR0	CORE CLOCK TO BUS CLOCK RATIO
84h	2.5 x
81h	3.0 x
85h	3.5 x
82h	4.0 x
86h	4.5 x
83h	5.0 x
87h	5.5 x
88h	6.0 x
8Ah	6.5 x
89h	7.0 x
8Bh	7.5 x

3. Cyrix III Configuration Register Index Assignments

On-chip configuration registers are used to control the on-chip cache, system management mode and other Cyrix III unique features.

3.1 Accessing a Configuration Register

Access to the configuration registers is achieved by writing the index of the register to I/O port 22h. I/O port 23h is then used for data transfer. Each I/O port 23h data transfer must be preceded by an I/O port 22h register index selection, otherwise the second and later I/O port 23h operations are directed off-chip and produce external I/O cycles. Reads of I/O port 22h are always directed off-chip. Appendix D contains example code for accessing the Cyrix III configuration registers.

3.2 Cyrix III Configuration Register Index Assignments

The table on the following page lists the Cyrix III configuration register index assignments. After reset, configuration registers with indexes C0-CFh and FC-FFh are accessible. In order to prevent potential conflicts with other devices which may use ports 22 and 23h to access their registers, the remaining registers (indexes 00-BFh, D0-FBh) are accessible only if the MAPEN(3-0) bits in CCR3 are set to 1h. With MAPEN(3-0) set to 1h, any access to an index in the 00-FFh range does not create external I/O bus cycles. Registers with indexes C0-CFh, FC-FFh are accessible regardless of the state of the MAPEN bits. If the register index number is outside the C0-CFh or FE-FFh ranges, and MAPEN is set to 0h, external I/O bus cycles occur. The table on the next page lists the MAPEN values required to access each Cyrix III configuration register. The configuration registers are described in more detail in the following sections.

Table 18. Configuration Register Index Assignments

REGISTER INDEX	REGISTER NAME	ACRONYM	WIDTH (BITS)	MAPEN(3-0)
00h-BFh	Reserved	—	—	—
C0h	Configuration Control 0	CCR0	8	Don't care
C1h	Configuration Control 1	CCR1	8	Don't care
C2h	Configuration Control 2	CCR2	8	Don't care
C3h	Configuration Control 3	CCR3	8	Don't care
E8h	Configuration Control 4	CCR4	8	1h
E9h	Configuration Control 5	CCR5	8	1h
EAh	Configuration Control 6	CCR6	8	1h
EBh	Configuration Control 7	CCR7	8	1h
C4h-C6h	Address Region 0	ARR0	24	Don't care
C7h-C9h	Address Region 1	ARR1	24	Don't care
CAh-CCh	Address Region 2	ARR2	24	Don't care
CDh-CFh	Address Region 3	ARR3	24	Don't care
D0h-D2h	Address Region 4	ARR4	24	1h
D3h-D5h	Address Region 5	ARR5	24	1h
D6h-D8h	Address Region 6	ARR6	24	1h
D9h-DBh	Address Region 7	ARR7	24	1h
A4h-A6h	Address Region 8	ARR8	24	x01x
A7h-A9h	Address Region 9	ARR9	24	x01x
AAh-ACh	Address Region A	ARRA	24	x01x
ADh-AFh	Address Region B	ARRB	24	x01x
D0h-D2h	Address Region C	ARRC	24	x01x
D6h-D8h	Address Region D	ARRD	24	x01x
DCh	Region Configuration 0	RCR0	8	1h
DDh	Region Configuration 1	RCR1	8	1h
DEh	Region Configuration 2	RCR2	8	1h
DFh	Region Configuration 3	RCR3	8	1h
E0h	Region Configuration 4	RCR4	8	1h
E1h	Region Configuration 5	RCR5	8	1h
E2h	Region Configuration 6	RCR6	8	1h
E3h	Region Configuration 7	RCR7	8	1h
DCh	Region Configuration 8	RCR8	8	x01x
DDh	Region Configuration 9	RCR9	8	x01x
DEh	Region Configuration A	RCRA	8	x01x

Table 18. Configuration Register Index Assignments

DFh	Region Configuration B	RCRB	8	x01x
E0h	Region Configuration C	RCRC	8	x01x
E1h	Region Configuration D	RCRD	8	x01x
E4h-E7h	Reserved	—	—	—
EBh-FAh	Reserved	—	—	—
FBh	Device Identification 2	DIR2	8	1h
FCh	Device Identification 3	DIR3	8	1h
FDh	Device Identification 4	DIR4	8	1h
FEh	Device Identification 0	DIR0	8	Don't care
FFh	Device Identification 1	DIR1	8	Don't care
48h	Bus Configuration Register 1	BCR1	8	0100
49h	Bus Configuration Register 2	BCR2	8	0100
41h	L2 Configuration Register 1	LCR1	8	0100
20h	Table Walk Register 0	TWR0	8	0001

The Cyrix III configuration registers can be grouped into five areas:

- Configuration Control Registers (CCRs)
- Address Region Registers (ARRs)
- Region Control Registers (RCRs)
- Device Identification Registers (DIRs)
- Cache and Bus Configuration Registers (BCRs)

CCR bits independently control Cyrix III features. ARRs and RCRs define regions of memory with specific attributes. DIRs are used for CPU detection as discussed earlier in Chapter 3. All bits in the configuration registers are initialized to zero following reset unless specified otherwise. The appropriate configuration register bit settings vary depending on system design. Optimal settings recommended for a typical PC environment are discussed in Chapter 5.

3.3 Configuration Control Registers (CCR0-7)

There are seven CCRs in the Cyrix III which control the cache, power management and other unique features. The following paragraphs describe the CCRs and associated bit definitions in detail.

3.3.1 Configuration Control Register 0 (CCR0)

Table 19. Configuration Control Register 0 (CCR0)

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

Table 20. CCR0 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
NC1	1	If = 1, designates 640KBytes -1MByte address region as non-cacheable. If = 0, designates 640KBytes -1MByte address region as cacheable.

3.3.2 Configuration Control Register 1 (CCR1)

Table 21. Configuration Control Register 1 (CCR1)

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

Table 22. CCR1 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
SM3	7	If = 1, designates Address Region Register 3 as SMM address space.

3.3.3 Configuration Control Register 2 (CCR2)

Table 23. Configuration Control Register 2 (CCR2)

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

Table 24. CCR2 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
WPR1	4	If = 1, designates that any cacheable accesses in the 640 KBytes-1MByte address region are write-protected. With WPR1=1, any attempted write to this range will not update the internal cache.
SUSP_HLT	3	If = 1, execution of the HLT instruction causes the CPU to enter low power suspend mode. This bit should be used with caution since the CPU must recognize and service an INTR, NMI or SMI to exit the “HLT initiated” suspend mode.
LOCK_NW	2	If = 1, the NW bit in CR0 becomes read only and the CPU ignores any writes to this bit.

3.3.4 Configuration Control Register 3 (CCR3)

Table 25. Configuration Control Register 3 (CCR3)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
MAPEN				Reserved	Reserved	NMI_EN	SMI_LOCK

Table 26. CCR3 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
MAPEN	7-4	Can be set to access various registers for read and write according to table 18.
NMI_EN	1	If = 1, NMI interrupt is recognized while in SMM. This bit should only be set while in SMM, after the appropriate NMI interrupt service routine has been setup.
SMI_LOCK	0	If = 1, the CPU prevents modification of the following SMM configuration bits, except when operating in an SMM service routine: CCR1 USE_SMI, SMAC, SM3 CCR3 NMI_EN ARR3 Starting address and block size. Once set, the SMI_LOCK bit can only be cleared by asserting the RESET pin.

3.3.5 Configuration Control Register 4 (CCR4)

Table 27. Configuration Control Register 4 (CCR4)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
CPUID	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

Table 28. CCR4 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
CPUID	7	If = 1, bit 21 of the EFLAG register is write/readable and the CPUID instruction will execute normally. If = 0, bit 21 of the EFLAG register is not write/readable and the CPUID instruction is an invalid opcode.

3.3.6 Configuration Control Register 5 (CCR5)

Table 29. Configuration Control Register 5 (CCR5)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Reserved	Reserved	ARREN	Reserved	Reserved	Reserved	Reserved	WT_ALLOC

Table 30. CCR5 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
ARREN	5	If = 1, enables all Address Region Registers (ARRs). If clear, disables the ARR registers. If SM3 is set, ARR3 is enabled regardless of the ARREN setting.
WT_ALLOC	0	If = 1, new cache lines are allocated for both read misses and write misses. If = 0, new cache lines are only allocated on read misses.

3.3.7 Configuration Control Register 6 (CCR6)

Table 31. Configuration Control Register 6 (CCR6)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	WP_ARR3	SMM_MODE

Table 32. CCR6 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
WP_ARR3	1	If = 1: Memory region defined by ARR3 is write protected when operating outside of SMM mode. If = 0: Disable write protection for memory region defined by ARR3. Reset State = 0.
SMM_MODE	0	If = 1: Enables Cyrix Enhanced SMM mode. If = 0: Disables Cyrix Enhanced SMM mode.

3.3.8 Configuration Control Register 7 (CCR7)

Table 33. Configuration Control Register 7 (CCR7)

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

Table 34. CCR7 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
3DNOW_EN	4	If = 1: 3DNOW instruction set is enabled If = 0: 3DNOW instruction set it disabled

3.3.9 Table Walk Register 0 (TWR0)

Table 35. Table Walk Register 0 (TWR0)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Reserved	Reserved	Reserved	CACHE_TE	Reserved	Reserved	Reserved	Reserved

Table 36. TWR0 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
CACHE_TE	4	If = 1: Cache Page Table Entries and Directory Table Entries. This is a performance enhancement If = 0:Do not cache table entries. Performance will be less.

3.4 Address Region Registers (ARR0-7)

The Address Region Registers (ARRs) are used to define up to eight memory address regions. Each ARR has three 8-bit registers associated with it which define the region starting address and block size. The Table "ARRx Index Assignments" below shows the general format for each ARR and lists the index assignments for the ARR's starting address and block size.

The region starting address is defined by the upper 12 bits of the physical address. The region size is defined by the BSIZE(3-0) bits as shown in the Table "BSIZE (3-0) Bit Definitions" on the next page. The BIOS and/or its utilities should allow definition of all ARRs. There is one restriction when defining the address regions using the ARRs. The region starting address must be on a block size boundary. For example, a 128KByte block is allowed to have a starting address of 0KBytes, 128KBytes, 256KBytes, and so on.

Table 37. ARRx Index Assignments

Address Region regisTer	Starting Address			Region Block Size
	A31-A24	A23-A16	A15-A12	BSIZE(3-0)
	Bits (7-0)	Bits (7-0)	Bits (7-4)	Bits (3-0)
ARR0	C4h	C5h	C6h	
ARR1	C7h	C8h	C9h	
ARR2	CAh	CBh	CCh	
ARR3	CDh	CEh	CFh	
ARR4	D0h	D1h	D2h	
ARR5	D3h	D4h	D5h	
ARR6	D6h	D7h	D8h	
ARR7	D9h	DAh	DBh	
ARR8	A4h	A5h	A6h	
ARR9	A7h	A8h	A9h	
ARRA	AAh	ABh	ACh	
ARRB	ADh	AEh	AFh	
ARRC	D0h	D1h	D2h	
ARRD	D3h	D4h	D5h	

Table 38. BSIZE (3-0) Bit Definitions

BSIZE(3-0)	ARR(0-6) REGION SIZE	ARR(7-D) REGION SIZE
0h	Disabled	Disabled
1h	4 KBytes	256 KBytes
2h	8 KBytes	512 KBytes
3h	16 KBytes	1 MByte
4h	32 KBytes	2 MBytes
5h	64 KBytes	4 MBytes
6h	128 KBytes	8 MBytes
7h	256 KBytes	16 MBytes
8h	512 KBytes	32 MBytes
9h	1 MByte	64 MBytes
Ah	2 MBytes	128 MBytes
Bh	4 MBytes	256 MBytes
Ch	8 MBytes	512 MBytes
Dh	16 MBytes	1 GBytes
Eh	32 MBytes	2 GBytes
Fh	4 GBytes	4 GBytes

3.5 Region Control Registers (RCR0-D)

The RCRs are used to define attributes, or characteristics, for each of the regions defined by the ARRs. Each ARR has a corresponding RCR with the general format shown below.

New to the Cyrix III is the Invert Region feature. This feature is controlled by the INV_RGN bit of the Region Control Registers.

If the INV_RGN bit is set, the controls specified in the RCR (RCD, WT, WG, WP) will be applied to all memory addresses *outside* the region specified in the corresponding ARR.

If the INV_RGN bit is cleared, the Cyrix III functions identically to the 6x86 and MII (the controls specified in the RCR will be applied to all memory addresses *inside* the region specified by the corresponding ARR).

The INV_RGN bit is defined for RCR(0-6) only. 6x86 Weak Write Ordering and Local Bus Access features have been eliminated on the Cyrix III. Therefore, bit 5 and bit 1 are reserved bits for the Cyrix III.

Table 39. RCR Bit Definitions

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Reserved	INV_RGN	Reserved	WT	WG	WP	Reserved	RCD/RCE

Note: RCD is defined for RCR0-RCR6. and RCR8-RCRD. RCE is defined for RCR7 only.

Table 40. RCR Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
RCD	0	Applicable to RCR(0-6) only. If set, the address region specified by the corresponding ARR is non-cacheable.
RCE	0	Applicable to RCR7 only. If set, the address region specified by ARR7 is cacheable and implies that address space outside of the region specified by ARR7 is non-cacheable.
WP	2	If set, **write protect** is enabled for the corresponding region.
WG	3	If set, **write gathering** is enabled for the corresponding region.
WT	4	If set, **write through** caching is enabled for the corresponding region.
INV_RGN	6	Applicable to RCR(0-6) only. If set, apply controls specified in RCR to all memory addresses outside the region specified in the corresponding ARR.

3.5.1 Detailed Description of RCR Attributes

Region Cache Disable (RCD)

Setting RCD=1 defines the corresponding address region as non-cacheable. RCD prevents caching of any access within the specified region. Additionally, RCD implies that high performance features are disabled for accesses within the specified address region.

Region Cache Enable (RCE)

Setting RCE=1 defines the corresponding address region as cacheable. RCE is applicable to ARR7 only. RCE in combination with ARR7, is intended to define the Main Memory Region. All memory outside ARR7 is non-cacheable when RCE is set. This is intended to define all unused memory space as non-cacheable.

Write Protect (WP)

Setting WP=1 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. This is useful for caching of shadowed ROM data.

Write Gathering (WG)

Setting WG=1 enables write gathering for the corresponding address region. With WG enabled, multiple byte, word or dword writes to sequential addresses that would normally occur as individual write cycles are combined and issued as a single write cycle. WG improves bus utilization and should be used for memory regions that are not sensitive to the “gathering.” WG can be enabled for both cacheable and non-cacheable regions.

Write Through (WT)

Setting WT=1 defines the corresponding 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.

Note that only one of these properties may be set per region. These are mutually exclusive memory type definitions, not properties that can be combines. For example WP basically means read only, thus write gathering and write though are irrelevant. WT means update main memory immediately upon writes, thus this excludes write gathering. Write gathering also defines when to write out data onto the bus, thus violating normal write back cache mode.

3.5.2 Attributes for Accesses Outside Defined Regions

If an address is accessed that is not in a region defined by the ARRs and ARR7 is defined with RCE=1, the following conditions apply:

- The memory access is not cached.
- Writes are not gathered.
- Strong write ordering occurs.

3.5.3 Attributes for Accesses in Overlapped Regions

If two defined address regions overlap (including NC1 and LBR1) and conflicting attributes are specified, the following attributes take precedence:

- Write-back is disabled.
- Writes are not gathered.
- Strong write ordering occurs.
- The overlapping regions are non-cacheable.

Since the CCR0 bit NC1 affects cacheability, a potential exists for conflict with the ARR7 main memory region which also affects cacheability. This overlap in address regions causes a conflict in cacheability. In this case, NC1 takes precedence over the ARR7/RCE setting because non-cacheability always takes precedence. For example, for the following settings:

- NC1=1
- ARR7 = 0-16 MBytes
- RCR7 bit RCE = 1

The Cyrix III caches accesses as shown in the table below.

Table 41. Cacheability for Example 1

ADDRESS REGION	CACHEABLE	COMMENTS
0 to 640 KBytes	Yes	ARR7/RCE setting.
640 KBytes- 1 MByte	No	NC1 takes precedence over ARR7/ RCE setting.
1 MByte - 16 MBytes	Yes	ARR7/RCE setting.
16 MBytes - 4 GBytes	No	Default setting.

3.6 BIOS Core-to-Bus Clock Ratio Configuration Registers

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

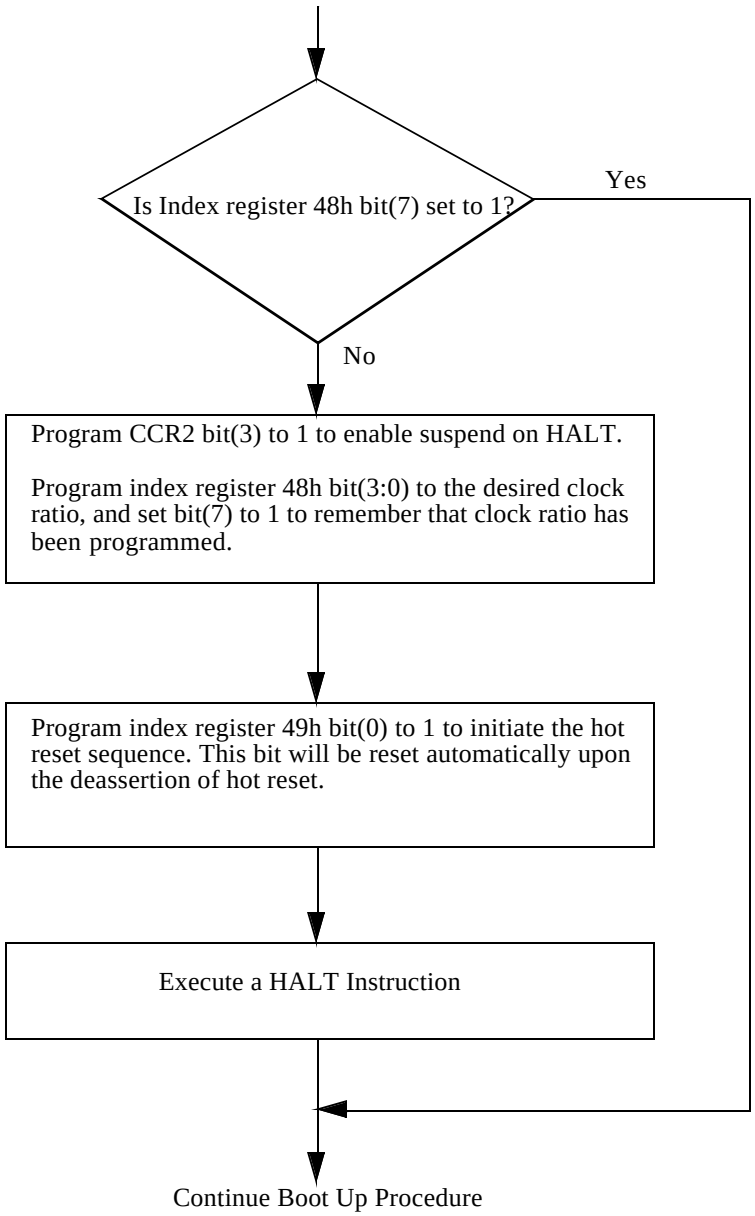
BIT	NAME	DESCRIPTION
7	HOTRST_TRIGGERED	A read/write bit used to indicate to the BIOS whether hot reset has been triggered or not. Default value is 0.
6	Reserved.	0
5:4	BSEL[1- 0]	Indicate the P6 bus speed.
3:0	BIOS_CLKRATIO[3-0]	Core-to-bus clock ratio as described below.

BITS (3:0)	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

Index: 49h
Default Value:
Access: Read/Write
MAPEN: 0100b

BIT	NAME	DESCRIPTION
7:1	Reserved	
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.

These registers may be used for motherboards that do not have jumpers for selecting the clock ratios. The following flowchart illustrates the procedure to set the clock multiplier through BIOS.


Is Index register 48h bit(7) set to 1?
Yes
No
Program CCR2 bit(3) to 1 to enable suspend on HALT.
Program index register 48h bit(3:0) to the desired clock ratio, and set bit(7) to 1 to remember that clock ratio has been programmed.
Program index register 49h bit(0) to 1 to initiate the hot reset sequence. This bit will be reset automatically upon the deassertion of hot reset.
Execute a HALT Instruction
Continue Boot Up Procedure

3.7 L2 Cache Control

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.

4. PC Memory Model

The table below defines the allowable attributes for a typical PC memory model. Actual recommended configuration register settings for a typical PC system are listed in AppendixF.

Table 42. PC Memory Model

ADDRESS SPACE	ADDRESS RANGE	CACHEABLE	WRITE PROTECTED	WRITE GATH-ERED	WRITE-THROUGH	NOTES
DOS Area	0-9 FFFFh	Yes	No	No	No	
Video Buffer	A 0000-B FFFFh	No	No	Yes	No	Note 1
Video ROM	C 0000-C 7FFFh	Yes	Yes	No	No	Note 2
Expansion Card/ROM Area	C 8000h-D FFFFh	No	No	No	No	
System ROM	E 0000h-F FFFFh	Yes	Yes	No	No	Note 2
Extended Memory	10 0000h-Top of Main Memory	Yes	No	Yes	No	
Unused/PCI MMIO	Top of Main Memory-FFFF FFFFh	No	No	No	No	Note 3

Notes 1: Video Buffer Area

A non-cacheable region must be used to enforce strong cycle ordering in this area and to prevent caching of Video RAM. The Video RAM area is sensitive to bus cycle ordering. The VGA controller can perform logical operations which depend on strong cycle ordering (found in Windows 3.1 code). To guarantee that the Cyrix III performs strong cycle ordering, a non-cacheable area must be established to cover the Video RAM area.

Video performance is greatly enhanced by gathering writes to Video RAM. For example, video performance benchmarks have been found to use REP STOSW instructions that would normally execute as a series of sequential 16-bit write cycles. With WG enabled, groups of four 16-bit write cycles are reduced to a single 64-bit write cycle.

Note 2: Video ROM and System ROM

Caching of the Video and System ROM areas is permitted, but is normally non-cacheable because NC1 is set. If these areas are cached, they must be cached as write-through regions. Cyrix III system benchmarking in a Windows environment has shown no benefit to caching these ROM areas. Therefore, it is recommended that these areas be set as non-cacheable using the NC1 bit in CCR0.

Note 3: Top of Main Memory-FFFF FFFFh (Unused/PCI Memory Space)

Unused/PCI Memory Space immediately above physical main memory must be defined as non-cacheable to ensure proper operation of memory sizing software routines and to guarantee strong cycle ordering. Memory discovery routines must occur with cache disabled to prevent read sourcing from the write buffers. Also, PCI memory mapped I/O cards that may exist in this address region may contain control registers or FIFOs that depend on strong cycle ordering.

The appropriate non-cacheable region must be established using ARR7. For example, if 32 MBytes (000 0000h-1FF FFFFh) are installed in the system, a non-cacheable region must begin at the 32 MByte boundary (200 0000h) and extend through the top of the address space (FFFF FFFFh). This is accomplished by using ARR7 (Base = 0000 0000h, BSize = 32 MBytes) in combination with RCE=1.

4.1 *General Recommendations*

4.1.1 Main Memory

Memory discovery routines should always be executed with the L1 cache disabled. By default, L1 caching is globally disabled following reset because the CD bit in Control Register 0 (CR0) is set. Always ensure the L1 cache is disabled by setting the CD bit in CR0 or by programming an ARR to “4 GByte cache disabled” before executing the memory discovery routine. Once BIOS completes memory discovery, ARR7 should be programmed with a base address of 000 0000h and with a “Size” equal to the amount of main memory that was detected.

The intent of ARR7 is to define a cacheable region for main memory and simultaneously define unused/PCI space as non-cacheable. More restrictive regions are intended to overlay the 640k to 1MByte area. Failure to program ARR7 with the correct amount of main memory can result in:

- Incorrect memory sizing by the operating system eventually resulting in failure,
- PCI devices not working correctly or causing the system to hang,
- Low performance if ARR7 is programmed with a smaller size than the actual amount of memory.

If the granularity selection in ARR7 does not accommodate the exact size of main memory, unused ARRs can be used to fill-in as non-cacheable regions. All unused/PCI memory space must always be set as non-cacheable.

4.1.2 BIOS Creation Utilities

BIOS creation utilities or setup screens must have the capability to easily define and modify the contents of the Cyrix III configuration registers. This allows OEMs and integrators to easily configure these register settings with the values appropriate for their system design.

4.2 Recommended Bit Settings

4.2.1 NC1

The NC1 bit in CCR0 controls the predefined non-cacheable region from 640K to 1MByte. The 640K to 1MByte region should be non-cacheable to prevent L1 caching of expansion cards using memory mapped I/O (MMIO). Setting NC1 also implies that the video BIOS and system BIOS are non-cacheable. Experiments with both the Cyrix III and Pentium CPUs have shown that performance is largely unchanged whether the video BIOS and system BIOS was cached or not. This assumes that a modern operating system was used and that the measurements are taken with a recent benchmark applications, such as WinStone95.

Recommended setting: NC1 = 1

4.2.2 LOCK_NW

Once set, LOCK_NW prohibits software from changing the NW bit in CR0. Since the definition of the NW bit is the same for both the Cyrix III and the Pentium, it is not necessary to set this bit.

Recommended setting: LOCK_NW = 0

4.2.3 WPR1

WPR1 forces cacheable accesses in the 640k to 1MByte address region to be write-protected. If NC1 is set (recommended setting), all caching is disabled from 640k to 1MByte and WPR1 is not required. However, if ROM areas within the 640k-1MByte address region are cached, WPR1 should be set to protect against errant self-modifying code.

Recommended setting: WPR1 = 0 unless ROM areas are cached

4.2.4 MAPEN

When set to 1h, the MAPEN bits allow access to all Cyrix III configuration registers including indices outside the C0h-CFh and FCh-FFh ranges. MAPEN should be set to 1h only to access specific configuration registers and then should be cleared immediately after the access is complete.

Recommended setting: MAPEN(3-0) = 0 except for specific configuration register accesses

4.2.5 CPUID

When set, the CPUID bit enables the CPUID instruction. By default, the CPUID instruction is enabled (CPUID = 1).

When enabled, the CPUID opcode is enabled and the CPUID bit in the EFLAGS can be modified. The CPUID instruction can then be called to inspect the type of CPU present.

When the CPUID instruction is disabled (CPUID = 0), the CPUID opcode 0FA2 causes an invalid opcode exception. Additionally, the CPUID bit in the EFLAGS register cannot be modified by software.

Recommended setting: CPUID = 1

4.2.6 WT_ALLOC

Write Allocate (WT_ALLOC) allows L1 cache write misses to cause a cache line allocation. This feature improves the L1 cache hit rate resulting in higher performance. Especially useful for Windows applications.

Recommended setting: WT_ALLOC = 1

4.2.7 ARREN

The ARREN bit enables or disables all eight ARRs. When ARREN is cleared (default), the ARRs can be safely programmed. Most systems will need to use at least one address region register (ARR). Therefore, ARREN should always be set after the ARRs and RCRs have been initialized.

Recommended setting: ARREN = 1 after initializing ARR0-ARR7, RCR0-RCR7

4.2.8 ARR7 and RCR7

Address Region 7 (ARR7) defines the Main Memory Region (MMR). This region specifies the amount of cacheable main memory and it's attributes. Once BIOS completes memory discovery, ARR7 should be programmed with a base address of 000 0000h and with a "Size" equal to the amount of main memory installed in the system. Memory accesses outside of this region are defined as non-cacheable to ensure compatibility with PCI devices.

Recommended settings:
ARR7 Base Addr= 0000 0000h
ARR7 Block Size= amount of main memory
RCR7 RCE = 0
RCR7 WL = 0
RCR7 WG = 1
RCR7 WT = 0

If the granularity selection in ARR7 does not accommodate the exact size of main memory, unused ARRs can be used to fill-in as non-cacheable regions (RCD = 1) as shown in the table below. All unused/PCI memory space must always be set as non-cacheable.

Table 43. ARR Settings for Various Main Memory Sizes

MEM	ARR7		ARR6		ARR5		ARR4	
SIZE (MB)	BASE (HEX)	SIZE (MB)	BASE (HEX)	SIZE (MB)	BASE (HEX)	SIZE (MB)	BASE (HEX)	SIZE (MB)
8	0	8						
16	0	16						
24	0	32	0180 0000	8				
32	0	32						
40	0	64	0300 0000	16	0280 0000	8		
48	0	64	0300 0000	16				
64	0	64						
72	0	128	0600 0000	32	0500 0000	16	0480 0000	8
80	0	128	0600 0000	32	0500 0000	16		
96	0	128	0600 0000	32				
128	0	128						
160	0	256	0E00 0000	32	0C00 0000	32	0A00 0000	32
192	0	256	0E00 0000	32	0C00 0000	32		
256	0	256						

4.2.9 Regions above Main Memory

Memory regions above main memory such as memory mapped i/o and write gathering or write combining regions for video memory can be setup using ARRs as well. The memory map table below gives an example of how to setup entire memory up to 4G to include physical memory, i/o space, write gathering regions for video, and uncacheable unused space.

ADDRESS SPACE	ARR	START	SIZE	RCR BIT(7) **RCD/E**	RCR BIT(2) **WP**	RCR BIT(3) **WG**	RCR BIT(4) **WT**
Main Memory	ARR7	0	Physical Memory	1	0	0	0
Memory mapped i/o space	ARR8	Physical Memory	i/o space size	1	0	0	0
Write gathering region	ARR9	i/o space top	Write Gathering region size	0	0	1	0
Uncacheable region	ARRA	Write Gathering region top	4G minus top of Write Gathering Region	1	0	0	0

4.2.10 SMM Features

The Cyrix III supports SMM mode through the use of the SMI# and SMIACT# pins, and a dedicated memory region for the SMM address space. SMM features must be enabled prior to servicing any SMI interrupts. The following paragraphs describe each of the SMM features and recommended settings.

SM3 and ARR3

Address Region Register 3 (ARR3) can be used to define the System Management Address Region (SMAR). Systems that use SMM features must use ARR3 to establish a base and limit for the SMM address space.

Only ARR3 can be used to establish the SMM region.

Typically, SMAR overlaps normal address space. RCR3 defines the attributes for both the SMM address region *and* the normal address space. If SMAR overlaps main memory, write gathering should be enabled for ARR3. If SMAR overlaps video memory, ARR3 should be set as non-cacheable and write gathering should be enabled.

NMI_EN

The NMI_EN bit allows NMI interrupts to occur within an SMI service routine. If this feature is enabled, the SMI service routine must guarantee that the IDT is initialized properly to allow the NMI to be serviced. Most systems do not require this feature.

SMI_LOCK

Once the SMM features are initialized in the configuration registers, they can be permanently locked using the SMI_LOCK bit. Locking the SMM related bits and registers prevents applications from tampering with these settings. Even if SMM is not implemented, setting SMI_LOCK in combination with SMAC=0 prevents software SMIs from occurring.

Once SMI_LOCK is set, it can only be cleared by a processor RESET. Consequently, setting SMI_LOCK makes system/BIOS/SMM debugging difficult. To alleviate this problem, SMI_LOCK must be implemented as a user selectable “Secure SMI (enable/disable)” feature in CMOS setup. If SMI_LOCK is not user selectable, it is recommended that SMI_LOCK = 0 to allow for system debug.

Suggested settings for systems not using SMM:

SM3	= 0
ARR3	= may be used as normal address region register
SMI_LOCK	= 0
NMI_EN	= 0

Suggested settings for systems using SMM:

SM3	= 1
ARR3 Base Addr	= as required
ARR3 Block Size	= as required
SMI_LOCK	= 0
NMI_EN	= 0

4.2.11 Power Management Features

SUSP_HALT

Suspend on Halt (SUSP_HLT) permits the CPU to enter a low power suspend mode when a HLT instruction is executed. Although this provides some power management capability, it is not optimal.

Suggested setting:

SUSP_HALT	= 1

5. Model Specific Registers

The Cyrix III contains four model specific registers (MSR0 - MSR3). These 64-bit registers are listed in the table below.

Table 44. Machine Specific Register

REGISTER DESCRIPTION	MSR ADDRESS	REGISTER
Time Stamp Counter (TSC)	10h	MSR10
Counter Event Selection and Control Register	11h	MSR11
Performance Counter #0	12h	MSR12
Performance Counter #1	13h	MSR13

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.

5.1 Time Stamp Counter

The Time Stamp Counter (TSC) Register (MSR10) 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 Cyrix III is suspend mode or shutdown.
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 Disable, (TSD) flag in CR4. When the TSD flag is 0, the RDTSC instruction can be executed at any privilege level. When the TSD flag is 1, the RDTSC instruction can only be executed at privilege level 0.

5.2 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 Control Register (MSR11). The perfor-

mance monitor counters use a continuous CPU core clock and will continue to count clock cycles even when the Cyrix III is in suspend mode or shutdown.

5.3 Performance Monitoring Counters 1 and 2

The 48-bit Performance Monitoring Counters (PMC) Registers (MSR12, MSR13) count events as specified by the counter event control register.
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 C4.
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.

5.4 Counter Event Control Register

Register MSR 11h 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 on page 46.

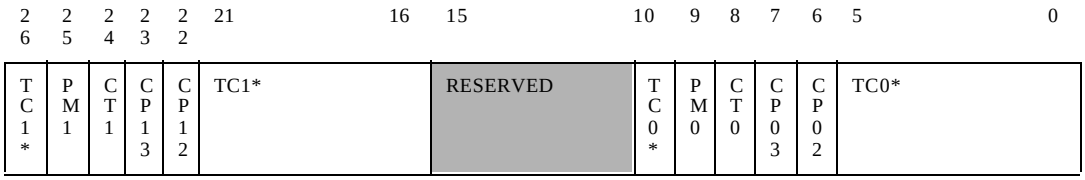
5.5 Performance Monitor Control

5.5.1 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.

5.5.2 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.



*Note: Split Fields

Table 45. Counter Event Control Register

Table 46. Counter Event Control Register Bit Definitions

BIT POSITION	NAME	DESCRIPTION
25	RSV	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
9	RSV	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

Note: Bits 10 - 15 are reserved.

5.5.3 Event Type and Description

The events that can be counted by the performance monitoring counters are listed in Figure 47. 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.

During occurrence type events, the PM pins are configured to indicate the counter has incremented The PM pins will then assert every time the counter increments in regards to an occurrence event. Under the same PM control, for a duration event the PM pin will stay asserted for the duration of the event.

Table 47. 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 Liens	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 pipes	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 pipes	Occurrence

Table 47. Event Type Register

NUMBER	COUNTER 0	COUNTER 1	DESCRIPTION	TYPE
17h	yes	yes	Instructions executed in Y pipe	Occurrence
18h	yes	yes	Clocks while bus cycles are in progress	Duration
19h	yes	yes	Pipe Stalled by full write buffers	Duration
1Ah	yes	yes	Pipe Stalled by waiting on data memory reads	Duration
1Bh	yes	yes	Pipe Stalled by writes to not-Modified or not-Exclusive cache lines.	Duration
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	Pipe 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 pipe	Occurrence
2Bh	no	yes	MMX Instruction Executed in Y pipe	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	
36	yes	no	Reserved	
36	no	yes	Reserved	
37	yes	no	Returns Predicted Incorrectly	Occurrence

Table 47. Event Type Register

NUMBER	COUNTER 0	COUNTER 1	DESCRIPTION	TYPE
37	no	yes	Return Predicted (Correctly and Incorrectly)	Occurrence
38	yes	no	MMX Instruction Multiply Unit Interlock	Duration
38	no	yes	MODV/MOVQ Store Stall Due to Previous Operation	Duration
39	yes	no	Returns	Occurrence
39	no	yes	RSB Overflows	Occurrence
3A	yes	no	BTB False Entries	Occurrence
3A	no	yes	BTB Miss Prediction on a Not-Taken Back	Occurrence
3B	yes	no	Number of Clock Stalled Due to Full Write Buffers While Executing	Duration
3B	no	yes	Stall on MMX Instruction Write to E or M Line	Duration
3C - 3Fh	yes	yes	Reserved	Duration
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	

6. Programming Model Differences

6.1 Instruction Set

The Cyrix III supports the Pentium Pro instruction set plus MMX instructions. Pentium extensions for virtual mode are not supported.

6.2 Configuring Internal Cyrix III Features

The Cyrix III supports configuring internal features through I/O ports.

6.3 INVD and WBINVD Instructions

The INVD and WBINVD instructions are used to invalidate the contents of the internal and external caches. The WBINVD instruction first writes back any modified lines in the cache and then invalidates the contents. It ensures that cache coherency with system memory is maintained regardless of the cache operating mode. Following invalidation of the internal cache, the CPU generates a special bus cycle to indicate that external caches should also write back modified data and invalidate their contents.

On the Cyrix III, the INVD functions differently from the WBINVD instruction. If the L2 is enabled, the Cyrix III will just invalidate the internal cache contents.

6.4 Control Register 0 (CR0) CD and NW Bits

The CPU's CR0 register contains, among other things, the CD and NW bits which are used to control the on-chip cache. CR0, like the other system level registers, is only accessible to programs running at the highest privilege level. The table on the following page lists the cache operating modes for all possible states of the CD and NW bits.

The CD and NW bits are set to one (cache disabled) after reset. For highest performance the cache should be enabled in write-back mode by clearing the CD and NW bits to 0. Sample code for enabling the cache is listed in Appendix E. To completely disable the cache, it is recommended that CD and NW be set to 1 followed by execution of the WBINVD instruction. The Cyrix III cache always accepts invalidation cycles even when the cache is disabled. Setting CD=0 and NW=1 causes a General Protection fault on the Pentium, but is allowed on the Cyrix III to globally enable write-through caching.

Table 48. Cache Operating Modes

CD	NW	OPERATING MODES
1	1	Cache disabled. Read hits access the cache. Read misses do not cause line fills. Write hits update the cache and system memory. Write hits change exclusive lines to modified. Shared lines remain shared after write hit. Write misses access memory. Inquiry and invalidation cycles are allowed. System memory coherency maintained.
1	0	Cache disabled. Read hits access the cache. Read misses do not cause line fills. Write hits update the cache. Only write hits to shared lines and write misses update system memory. Write misses access memory. Inquiry and invalidation cycles are allowed. System memory coherency maintained.
0	1	Cache enabled in Write-through mode. Read hits access the cache. Read misses may cause line fills. Write hits update the cache and system memory. Write misses access memory. Inquiry and invalidation cycles are allowed. System memory coherency maintained.
0	0	Cache enabled in Write-back mode. Read hits access the cache. Read misses may cause line fills. Write hits update the cache. Write misses access memory and may cause line fills if write allocation is enabled. Inquiry and invalidation cycles are allowed. System memory coherency maintained.

7. Appendixes

Appendix A.- Sample Code: Detecting a Cyrix CPU

```
assume  cs:_TEXT
public  _iscyrix
_TEXT   segment byte public 'CODE'
;*****************************************************************
;       Function: int iscyrix ()
;
;       Purpose:        Determine if Cyrix CPU is present
;       Technique:      Cyrix CPUs do not change flags where flags
;                                                       change in an undefined manner on other CPUs
;       Inputs:         none
;       Output:         ax == 1 Cyrix present, 0 if not
;*****************************************************************
_iscyrix    proc    near
                        .386
                        xor   ax, ax        ; clear ax
                        sahf              ; clear flags, bit 1 always=1 in flags
                        mov   ax, 5
                        mov   bx, 2
                        div   bl          ; operation that doesn't change flags
                        lahf              ; get flags
                        cmp   ah, 2       ; check for change in flags
                        jne   not_cyrix   ; flags changed, therefore NOT CYRIX
                        mov   ax, 1       ; TRUE Cyrix CPU
                        jmp   done

not_cyrix:
                        mov  ax, 0        ; FALSE NON-Cyrix CPU
done:
                        ret
_iscyrix    endp
_TEXT ends
end
```

Appendix B. Sample Code: Determining CPU MHz

```
assume  cs:_TEXT
public _cpu_speed
_TEXT segment para public 'CODE'

comment~
*****************************************************************************
 Function:     unsigned long _cpu_speed( unsigned int )
               "C" style caller
 Purpose:      calculate elapsed time req'd to complete a loop of IDIVs

 Technique:    Use the PC's high resolution timer/counter chip (8254)
               to measure elapsed time of a software loop consisting
               of the IDIV and LOOP instruction.
 Definitions:  The 8254 receives a 1.19318MHz clock (0.8380966 usec).
               One "tick" is equal to one rising clock edge applied
                     to the 8254 clock input.

 Inputs:       ax = no. of loops for cpu_speed_loop
 Returns:      ax = no. of 1.19318MHz clk ticks req'd to complete a loop
               dx = state of 8254 out pin
*****************************************************************************~
PortB           EQU     061h
Timer_Ctrl_Reg  EQU     043h
Timer_2_Data    EQU     042h
stk$dx          EQU     10              ;dx register offset
stk$ax          EQU     14              ;dx register offset
stack$ax        EQU     [bp]+stk$ax
stack$dx        EQU     [bp]+stk$dx
Loop_Count      EQU     [bp+16]+4

.386p

_cpu_speed      proc near
        pushf                   ;save interrupt flag
        pusha                   ;pushes 16 bytes on stack
        mov     bp,sp           ;init base ptr

        cli                     ;disable interrupts

;-------disable clock to timer/counter 2
        in      al, PortB
        and     al, 0feh
        out     80h,al          ;I/O recovery time
        out     PortB, al
        mov     di, ax

;-------initialize the 8254 counter to "0", known value
        mov     al,0b0h
        out     Timer_Ctrl_Reg, al      ;control word to set channel 2 count
        out     80h,al          ;I/O recovery time
        mov     al,0ffh
        out     Timer_2_Data, al        ;init count to 0, lsb
        out     80h,al          ;I/O recovery time
```

```
        out     Timer_2_Data, al       ;init count to 0, msb

;-------get the number of loops from the caller's stack
        mov     cx,Loop_Count          ;loop count

;-------load dividend & divisor, clk count for IDIV depend on operands!
        xor     edx,edx                ;dividend EDX:EAX
        xor     eax,eax
        mov     ebx,1                  ;divisor

;-------enable the timer/counter's clock. Begin timed portion of test!
        xchg    ax, di                 ;save ax for moment
        or      al, 1
        out     PortB, al              ;enable timer/counter 2 clk
        xchg    ax, di                 ;restore ax

;-------this is the core loop.
        ALIGN 16
cpu_speed_loop:
        idiv    ebx
        idiv    ebx
        idiv    ebx
        idiv    ebx
        idiv    ebx
        loop    cpu_speed_loop

;-------disable the timer/counter's clk.  End timed portion of test!
        mov     ax, di
        and     al, 0FEH
        out     PortB, al

;-------send latch status command to the timer/counter
        mov     al, 0c8h               ;latch status and count
        out     Timer_Ctrl_Reg, al
        out     80h,al                 ;I/O recovery time

;-------read status byte, and count value "ticks" from the timer/cntr
        in      al, Timer_2_Data       ;read status
        out     80h,al                 ;I/O recovery time
        mov     dl, al
        and     dx, 080h
        shr     dx, 7

        in      al, Timer_2_Data       ;read LSB
        out     80h,al                 ;I/O recovery time
        mov     bl, al
        in      al, Timer_2_Data       ;read MSB
        out     80h,al                 ;I/O recovery time
        mov     bh, al

        not     bx                     ;invert count

;-------send command to clear the timer/counter
        mov     al, 0b6h
        out     Timer_Ctrl_Reg, al     ;clear channel 2 count
        out     80h,al                 ;I/O recovery time
        xor     al, al
```

```
        out     Timer_2_Data, al        ;set count to 0, lsb
        out     80h,al                  ;I/O recovery time
        out     Timer_2_Data, al        ;set count to 0, msb

;-------put return values on the stack for the caller
        mov     [bp+stk$ax], bx
        mov     [bp+stk$dx], dx

        popa
        popf                            ;restores interrupt flag
        ret
_cpu_speed      endp

.8086
_TEXT   ENDS
END
```

Appendix C. Example CPU Type and Frequency Detection Program

```
/* ************************************************************************
  function:  main()                                WCP 8/22/95
  Purpose:   a driver program to demonstrate:
             CPU detection
             CPU core frequency in Mhz.
  Returns:   0 if successful

  Required source code modules
    m1_stat.c                  main() module (this file)
    id.asm               cpu identification code
    clock.asm              cpu timing loop

  Compile and Link instructions for Borland C++ or equivalent:
     bcc m1_stat.c id.asm clock.asm
************************************************************************** */
/* include directives */
  #include <stdio.h>

/* constants */
  #define TTPS         1193182   //high speed Timer Ticks per second in Mhz
  #define MHZ          1000000   //number of clocks in 1 Mhz
  #define LOOP_COUNT    0x2000    //core loop iterations
  #define RUNS         10        //number of runs to average
  #define DIVS         5         //# of IDIV instructions in the core loop
  #define Cyrix III_IDIV_CLKS  17        //known clock counts for Cyrix III
  #define Cyrix III_LOOP_CLKS  1
  #define P54_IDIV_CLKS 46        //known clock counts for P54
  #define P54_LOOP_CLKS 7

/* prototypes */
  unsigned int  iscyrix( void );                  //detects cyrix cpu
  unsigned long cpu_speed( unsigned int );         //core timing loop

 main(){

 /* declarations */
  unsigned char     uc_cyrix_cpu = 0;              //Cyrix cpu? 0=no, 1=yes
  unsigned int      i_runs = 0;                   //number of runs to avg
  unsigned int      ui_idiv, ui_loop = 0;          //instruction clk counts
  unsigned long     ul_tt_cnt, ul_tt_sum = 0;       //timer tick counts, sum
  unsigned int      ui_core_loop_cntr = LOOP_COUNT; //core loop iterations
  float             f_mtt = 0;                    //measured timer ticks
  float             f_total_core_clks = 0;         //calculated core clocks
  float             f_total_time = 0;             //measured time
  float             f_mhz = 0;                     //mhz

/* ********** determine if Cyrix CPU is present ************** */

  //detect if Cyrix CPU is present
  uc_cyrix_cpu = iscyrix();                       //1=cyrix, 0=non-cyrix

  //display a msg
  if(uc_cyrix_cpu) printf("\nCyrix CPU present! ");
```

```
    else printf("\nCyrix CPU not present! ");

/* ********************* determine CPU Mhz ********************* */

  //count # of hi speed "timer ticks" to complete several runs of core loop
  for (i_runs = 0 ; i_runs < RUNS ; i_runs++) {
   ul_tt_cnt = cpu_speed( ui_core_loop_cntr );
   ul_tt_sum += ul_tt_cnt;                    //sum them all together
  }//end for

  //compute the avg number of high speed "timer ticks" for the several runs
  f_mtt = ul_tt_sum / RUNS;                   //compute the average

  //initialize variables with the "known" clock counts for a Cyrix III or P54
  if(uc_cyrix_cpu)ui_idiv=Cyrix III_IDIV_CLKS; else ui_idiv=P54_IDIV_CLKS;
  if(uc_cyrix_cpu)ui_loop=Cyrix III_LOOP_CLKS; else ui_loop=P54_LOOP_CLKS;

  //determine the total number of core clocks. (5 idivs are in the core loop)
  f_total_core_clks = (float)ui_core_loop_cntr * (ui_idiv * DIVS + ui_loop);

  //the time it took to complete the core loop can be determined by the
  //ratio of measured timer ticks(mtt) to timer ticks per second(TTPS).
  f_total_time = f_mtt / TTPS;

  //frequency can be found by the ratio of core clks to the total time.
  f_mhz = f_total_core_clks / f_total_time;
  f_mhz = f_mhz / MHZ;                        //convert to Mhz

  //display a msg
  printf("The core clock frequency is: %3.1f MHz\n\n",f_mhz);

  return(0);

 } //end main
```

Appendix D.- Sample Code: Programming Cyrix III Configuration Registers

Reading/Writing Configuration Registers

Sample code for setting NC1=1 in CCR0.

```
pushf                                   ;save the if flag
cli                                     ;disable interrupts
mov     al, 0c0h                ;set index for CCR0
out     22h, al                 ;select CCR0 register
in      al, 23h                 ;READ current CCR0 valueREAD

mov     ah, al
or      ah, 2h                  ;MODIFY, set NC1 bitMODIFY

mov     al, 0c0h                ;set index for CCR0
out     22h, al                 ;select CCR0 register
mov     al, ah
out     23h,al                  ;WRITE new value to CCR0WRITE
popf                                    ;restore if flag
```

Setting MAPEN

Sample code for setting MAPEN=1 in CCR3 to allow access to all the configuration registers.

```
pushf                                   ;save the if flag
cli                                     ;disable interrupts
mov     al, 0c3h                ;set index for CCR3
out     22h, al                 ;select CCR3 register
in      al, 23h                 ;current CCR3 valueREAD

mov     ah, al
and     ah,0Fh                  ;clear upper nibble of ah
or      ah, 10h                 ;MODIFY, set MAPEN(3-0)MODIFY

mov     al, 0c3h                ;set index for CCR3
out     22h, al                 ;select CCR3 register
mov     al, ah
out     23h,al                  ;WRITE new value to CCR3WRITE
popf                                    ;restore if flag
```

Appendix E. - Sample Code: Controlling the L1 Cache

Enabling the L1 Cache

;reading/writing CR0 is a privileged operation.

```
mov     eax, cr0
and   eax, 09fffffffh;clear the CD=0, NW=1 bits to enable write-back
mov   cr0, eax                          ;control register 0 write
wbinvd                                  ;optional, by flushing the L1 cache here it
                                        ;ensures the L1 cache is completely clean
```

Disabling the L1 Cache

```
mov     eax, cr0
or      eax, 060000000h ;set the CD=1, NW=1 bits to disable caching
mov     cr0, eax                        ;control register 0 write
wbinvd
```

Appendix F. - Example Configuration Register Settings

Below is an example of optimized Cyrix III settings for a 16 MByte system with PCI. Since SMI address space overlaps Video RAM at A0000h, WG is set to maintain the settings of the underlying region ARR0. If SMI address space overlapped system memory at 30000h, WG would be set. If SMI address space overlapped FLASH ROM at E0000h, only RCD would be set. Power management features are disabled in this example system.

Table 49. Configuration Register Settings Example

REGISTER	BIT(S)	SETTING	DESCRIPTION
CCR0	NC1	1	Disables caching from 640k-1MByte.
CCR1	SM3	1	Sets ARR3 as SMM address region.
CCR2	LOCK_NW	0	Locking NW bit not required.
	SUSP_HLT	0	Power management not required for this system.
	WPR1	0	ROM areas not cached, so WPR1 not required.
CCR3	SMI_LOCK	0	Locks SMI feature as initialized.
	NMI_EN	0	Servicing NMIs during SMI not required.
	MAPEN(3-0)	0	Always clear MAPEN for normal operation.
CCR4	CPUIDEN	1	Enables CPUID instruction.
CCR5	WT_ALLOC	1	Enables write allocation for performance.
	ARREN	1	Enables all ARRs.
ARR0	BASE ADDR	A0000h	Video buffer base address = A0000h.
	BLOCK SIZE	6h	Video buffer block size = 128KBytes.
RCR0	RCD	1	Caching disabled for compatibility. Caching also disabled via NC1.
	WP	0	
	WG	0	
	WT	0	
	INV_RGN	0	
ARR1	BASE ADDR	C0000h	Expansion Card/ ROM base address = C 0000h.
	BLOCK SIZE	7h	Expansion Card/ROM block size = 256KBytes.
RCR1	RCD	1	Caching disabled for compatibility. Caching also disabled via NC1.
	WP	0	
	WG	0	
	WT	0	
	INV_RGN	0	
ARR3	BASE ADDR	A0000h	SMM address region base address
	BLOCK SIZE	4h	SMM address space = 32 KBytes

Table 49. Configuration Register Settings Example

RCR3	RCD	1	Caching disabled due to overlap with video buffer.
	WP	0	
	WG	0	
	WT	0	
	INV_RGN	0	
ARR7	BASE ADDR	0h	Main memory base address = 0h.
	BLOCK SIZE	7h	Main memory size = 16 MBytes.
RCR7	RCE	1	Caching enabled for main memory.
	WP	0	
	WG		
	WT	0	
ARR(2,4-6)	BASE ADDR	0	ARR(2,4-6) disabled (default state).
	BLOCK SIZE	0	
RCR(2,4-6)	RCD	0	RCR(2,4-6) not required due to corresponding ARRs disabled (default state).
	WP	0	
	WG	0	
	WT	0	
	INV_RGN	0	

Printed in the United States of America

Via-Cyrix

2703 North Central Expressway

Richardson, Texas 75080-2010

United States of America