The IBM 6x86 Microprocessor BIOS Writer's Guide

IBM®

Application Note

Revision Summary: This document describes new 6x86 configuration

Introduction

Scope

This document is intended for 6x86 processor system BIOS writers. It is not a stand alone document but supplements other IBM and 6x86 processor documentation. This document includes recommendations for 6x86 processor detection and register settings.

The recommended settings are optimized for both performance and compatibility in a Windows95** Plug and Play (PnP), PCI-based system. Issues regarding optimum performance, CPU detection, chipset initialization, memory discovery, I/O recovery time, and others are described in detail.

6x86 Microprocessor Configuration Registers

The 6x86 processor uses on-chip configuration registers to control the on-chip cache, system management mode (SMM), device identification, and other 6x86 processor unique features. The on-chip registers are used to activate advanced features including performance enhancements. These performance features may be enabled "globally" in some cases, or by a user-defined address region. The flexible configuration of the 6x86 processor is intended to fit a wide variety of systems.

The Importance of Non-Cacheable Regions

The 6x86 processor has eight internal user-defined Address Region Registers. Among other attributes, the regions define cacheability vs. non-cacheability of the address regions. Using this cacheability information, the 6x86 processor 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. Although negating KEN# during a memory access on the bus prevents a cache line fill, it does not fully disable these performance features. In other words, negating KEN# is NOT equivalent to establishing a non-cacheable region in the 6x86 processor.

Detecting an IBM 6x86 CPU

6x86 processor detection must first be determined by the BIOS during Power-On Self Test using the method described below. Allowing 6x86 processor detection using CPUID at run-time is covered later.

It is important to note that the 6x86 processor's CPUID instruction is disabled following reset. Compatibility testing has found that some popular software does not correctly check the CPUID return values (e.g. Vendor Identification String and Family fields). This results in misidentification of CPU features which may cause a variety of runtime errors. By disabling the CPUID instruction, the 6x86 processor is assured to run code compatible with the 486 instruction set and programming model.

Detecting an IBM CPU

Since CPUID is disabled by default, it cannot be used to identify the 6x86 processor during BIOS POST. The correct method for detecting the presence of a 6x86 microprocessor during BIOS POST is a two step process. First, an IBM brand CPU must be detected. Second, the CPU's Device Identification Registers (DIRs) provide the CPU model and stepping information. Alternate methods of detecting the CPU are not recommended. These include detection algorithms using the value of EDX following reset, and other signature methods of determining if the CPU is an 8086, 80286, 80386, or 80486.

Detection of an IBM brand CPU is accomplished by checking the state of the undefined flags following execution of the divide instruction which divides 5 by 2 (5**4**2). The undefined flags in an IBM microprocessor remain unchanged following the divide. Alternate CPUs modify some of the undefined flags. Using operands other than 5 and 2 may prevent the algorithm from working correctly. Appendix A contains example code for detecting an IBM CPU using this method.

Detecting CPU Type and Stepping using DIRs

Once an IBM brand CPU is detected, the model and stepping level of the CPU can be determined. All IBM CPUs contain Device Identification Registers (DIRs) that exist as part of the configuration registers. The DIRs for all IBM CPUs exist at configuration register indexes 0FEh and 0FFh. (See *6x86 Processor Configuration Register Index Assignments* below.) Table 1 specifies the contents of the 6x86 processor DIRs.

DIR0 bits [7:3] = 00110h indicate a 6x86 processor is present, DIR0 bits [2:0] indicate the core-to-bus clock ratio, and DIR1 contains stepping information. Clock ratio information is provided to assist calculations in determining bus frequency once the CPU's core frequency has been calculated. Proper bus speed settings are critical to overall system performance..

Table 1. 6x86 Processor Device Identification Registers

REGISTER	Description	Bit Position	CONTENTS	CORE/BUS CLOCK RATIO
DIR0	CPU Model	7-0	30h or 32h 31h or 33h 35h or 37h 34h or 36h	1/1 2/1 4/1 3/1
DIR1	Device Stepping	7-0	TBD	--

Determining 6x86 Operating Frequency

Determining the operating frequency of the CPU is normally required for correct initialization of the system logic. Typically, a software timing loop with known instruction clock counts is timed using legacy hardware (the 8254 timer/counter circuits) within the PC. Once the operating frequency of the 6x86 processor's core is known, DIR0 bits (2:0) can be examined to calculate the bus operating frequency.

Careful selection of instructions and operands must be used to replicate the exact clock counts detailed in the Instruction Set Summary in the 6x86 processor Data Book. An example code sequence for determining the 6x86 processor's operating frequency is detailed in Appendix B and Appendix C. The core loop uses a series of five IDIV instructions within a LOOP instruction. IDIV was chosen because it is an exclusive instruction meaning that it executes in the 6x86 processor x pipeline with no other instruction in the y pipeline. This allows for more predictable execution times as compared to using non-exclusive instructions.

The 6x86 processor instruction clock count for IDIV varies from 17 to 45 clocks for a doubleword divide depending on the value of the operands. The code example in the appendixes uses "0" divided by "1" which takes only 17 clocks to complete. The LOOP instruction clock count is 1. Therefore, the overall clock count for the inner loop in this example is 86 clocks.

CPUID Instruction

The CPUID instruction is disabled following reset to improve compatibility with existing software. It can be enabled by setting the CPUIDEN bit in configuration register CCR4. It is recommend that all BIOS vendors include a CPUID enable/disable field in the CMOS setup to allow the end user to enable the CPUID instruction. CPUID must default to disabled and remain disabled unless enabled by the end user.

The CPUID instruction, opcode 0FA2h, provides information indicating IBM as the vendor and the family, model, stepping, and CPU features. Additional documentation on the CPUID instruction and how alternate CPUs execute this instruction can be found in the Pentium Processor User's Manual, Volume 3, page 25-62, Pentium Processor User's Manual, Volume 1, Page 3-7, and Intel's application note AP-485.

The EAX register provides the input value for the CPUID instruction. The EAX register is loaded with a value to indicate what information should be returned by the instruction.

Following execution of the CPUID instruction with an input value of "0" in EAX, the EAX, EBX, ECX and EDX registers contain the information shown in Figure 1. EAX contains the highest input value understood by the CPUID instruction, which for the 6x86 processor is "1". EBX, ECX and EDX contain the vendor identification string "CyrixInstead."

Following execution of the CPUID instruction with an input value of "1" loaded in EAX, EAX[15:0] will contain the value of 053x. EDX bit [0] contains a "1" indicating that an FPU is on chip.

```
switch (EAX)
        {
        case (0):
                EAX := 1
                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' */
                break
        case (1):
                EAX[7:0]  := 3xh
                EAX[15:8] := 05h
                EDX[0] := 1   /* 1=FPU Built In,0=No FPU */
                break
        default:
                EAX, EBX, ECX, EDX : Undefined
        }
```

Figure 1. Information Returned by CPUID Instruction

EDX Value following Reset

Some CPU detection algorithms may use the value of the CPU's EDX register following reset. The 6x86 processor's EDX register contains the data shown below following a reset initiated using the RESET pin:

EDX[31:16] = undefined
EDX[15:8] = 05h
EDX[7:0] = 3x

The value in EDX does not identify the vendor of the CPU. Therefore, EDX alone cannot be used to determine if an IBM CPU is present. However, BIOS should preserve the contents of EDX so that applications can use the EDX value when performing a user-defined shutdown (e.g. a reset performed with data 0Ah in the Shutdown Status byte (Index 0Fh) of the CMOS RAM Map).

6x86 Configuration Register Index Assignments

On-chip configuration registers are used to control the on-chip cache, system management mode and other 6x86 processor unique features.

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 6x86 processor configuration registers.

6x86 processor Configuration Register Index Assignments

Table 2 lists the 6x86 processor 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 0001b. With MAPEN(3-0) set to 0001b 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 FC-FFh ranges, and MAPEN is set to 0h, external I/O bus cycles occur. Table 2 lists the MAPEN values required to access each 6x86 processor configuration register. The configuration registers are described in more detail in the following sections

Register Index	Register Name	Acronym	Width (BITS)	MAPEN (3-0)
00h-BFh	Reserved	--	--	--
C0h	Configuration Control 0	CCR0	8	x
C1h	Configuration Control 1	CCR1	8	x
C2h	Configuration Control 2	CCR2	8	x
C3h	Configuration Control 3	CCR3	8	x
C4h-C6h	Address Region 0	ARR0	24	x
C7h-C9h	Address Region 1	ARR1	24	x
CAh-CCh	Address Region 2	ARR2	24	x
CDh-CFh	Address Region 3	ARR3	24	x
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

Register Index	Register Name	Acronym	Width (BITS)	MAPEN (3-0)
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
E4h-E7h	Reserved	--	--	--
E8h	Configuration Control 4	CCR4	8	1h
E9h	Configuration Control 5	CCR5	8	1h
EAh-FDh	Reserved	--	--	--
FEh	Device Identification 0	DIR0	8	x
FFh	Device Identification 1	DIR1	8	x

x = Don' t Care

Table 2 Configuration Register Index Assignments

The 6x86 processor configuration registers can be grouped into four areas:

- Configuration Control Registers (CCRs)
- Address Region Registers (ARRs)
- Region Control Registers (RCRs)
- Device Identification Registers (DIRs)

CCR bits independently control 6x86 processor features. ARRs and RCRs together define regions of memory with specific attributes. DIRs are used for CPU detection as discussed earlier in *Detecting an IBM 6x86 CPU.* 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. Recommendations for optimal settings for a typical PC environment are discussed in *Recommended 6x86 Configuration Register Settings.*

Configuration Control Registers (CCR0-5)

There are six CCRs in the 6x86 processor which control the cache, power management and other unique features. The following paragraphs describe the CCRs and associated bit definitions in detail.

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 3. CCR0 Bit Definitions

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

Configuration Control Register 1 (CCR1)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
SM3	Reserved	Reserved	NO_LOCK	Reserved	SMAC	USE_SMI	Reserved

Table 4. CCR1 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
SM3	7	If set, designates Address Region Register 3 for SMM address space.
NO_LOCK	4	If set, all bus cycles are issued with the LOCK# pin negated except page table accesses and interrupt acknowledge cycles. Interrupt acknowledge cycles are executed as locked cycles even though LOCK# is negated. With NO_LOCK set, previously noncacheable locked cycles are executed as unlocked cycles and therefore, may be cached. This results in higher CPU performance. See the section on Region Configuration Registers (RCR) for more information on eliminating locked CPU bus cycles only in specific address regions.
SMAC	2	If set, any access to addresses within the SMM address space access system management memory instead of main memory. SMI# input is ignored while SMAC is set. Setting SMAC=1 allows access to SMM memory without entering SMM. This is useful for initializing or testing SMM memory.
USE_SMI	1	If set, SMI# and SMIACT# pins are enabled. If clear, SMI# pin is ignored and SMIACT# pin is driven inactive.

Configuration Control Register 2 (CCR2)

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

Table 5. CCR2 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
USE_SUSP	7	If set, SUSP# and SUSPA# pins are enabled. If clear, SUSP# pin is ignored and SUSPA# pin floats. These pins should only be enabled if the external system logic (chipset) supports them.
WPR1	4	If set, designates that any cacheable accesses in the 640K bytes - 1M byte address region are write-protected. With WPR1=1, any attempted write to this range will not get issued to the external bus.
SUSP_HLT	3	If set, execution of the HLT instruction causes the CPU to enter low power suspend mode. This bit should be used cautiously since the CPU must recognize and service an INTR, NMI or SMI to exit the "HLT initiated" suspend mode.
LOCK_NW	2	If set, the NW bit in CR0 becomes read only and the CPU ignores any writes to this bit.

Configuration Control Register 3 (CCR3)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
MAPEN3	MAPEN2	MAPEN1	MAPEN0	Reserved	LINBRST	NMI_EN	SMI_LOCK

Table 6. CCR3 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
MAPEN(3-0)	7-4	If set to 0001b (1h), all configuration registers are accessible. If clear, only configuration registers with indexes C0-CFh, FEh and FFh are accessible.
LINBRST	2	If set, the IBM 6x86 will use a linear address sequence when performing burst cycles. If clear, the IBM 6x86 will use a “1+4” address sequence when performing burstcycles. The “1+4” address sequence is compatible with the Pentium’s burst address sequence.
NMI_EN	1	If set, 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 set, the CPU prevents modification of the the following SMM configuration bits, except when operating in SMM: 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.

Configuration Control Register 4 (CCR4)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
CPUIDEN	Reserved	Reserved	DTE_EN	Reserved	IORT(2-0)		

Table 7. CCR4 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
CPUIDEN	7	If set, bit 21 of the EFLAG register is write/readable and the CPUID instruction will execute normally. If clear, bit 21 of the EFLAG register is not write/readable and the CPUID instruction is an invalid opcode.
DTE_EN	4	If set, the Directory Table Entry cache is enabled.
IORT(2-0)	2-0	Specifies the minimum number of bus clocks between I/O accesses (I/O recovery time). The delay time is the minimum time from the beginning of one I/O cycle to the beginning of the next (i.e. ADS# to ADS# time). 0h = 1 clock 1h = 2 clocks 2h = 4 clocks 3h = 8 clocks 4h = 16 clocks 5h = 32 clocks (default value after RESET) 6h = 64 clocks 7h = no delay

Configuration Control Register 5 (CCR5)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Reserved	Reserved	ARREN	LBR1	Reserved	Reserved	Reserved	WT_ALLOC

Table 8. CCR5 Bit Definitions

BIT NAME	BIT NO.	DESCRIPTION
ARREN	5	If set, enables all Address Region Registers (ARR). If clear, disables the ARR registers . If SM3 is set, ARR3 is enabled regardless of the ARREN setting.
LBR1	4	If set, LBA# pin is asserted for all accesses to the 640K bytes - 1M byte address region. See the section ***Region Configuration Registers*** for more information on enabling/disabling LBA# for specific address regions.
WT_ALLOC	0	If set, new cache lines are allocated for both read misses and write misses. If clear, new cache lines are only allocated on read misses.

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. Table 9 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 Table 10. 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 128K byte block is allowed to have a starting address of 0K bytes, 128K bytes, 256K bytes, and so on.

Table 9. 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	

Table 10. BSIZE(3-0) Bit Definitions

BSIZE(3-0)	ARR0-ARR6 REGION SIZE	ARR7 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

Region Control Registers (RCR0-7)

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.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Reserved	Reserved	NLB	WT	WG	WL	WWO	RCD/RCE

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

Table 11. RCR Bit Definitions

Bit Name	Bit No.	Description
RCD	0	Applicable to RCR0-6 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 the address space outside of the region specified by ARR7 is non-cachable.
WWO	1	If set, weak write ordering is enabled for the corresponding region.
WL	2	If set, weak locking 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.
NLB	5	If set, LBA# is negated for the corresponding region.

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. Bus cycles issued to memory addresses within the specified region are single cycles with the CACHE# pin negated. If KEN# is asserted for a memory access within a region defined non-cacheable by RCD, the access is not cached.

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. If KEN# is negated for an access within a region defined cacheable by RCE, the access is not cached.

Weak Write Ordering (WWO)

Setting WWO=1 enables weak write ordering for the corresponding address region. Weak Write Ordering allows the 6x86 processor to retire writes out of sequence to the internal cache only. External write cycles always occur in sequence (strongly ordered). WWO is only applicable to memory regions that have been cached and designated as write-back. WWO should never be enabled for memory mapped I/O.

Weak Locking (WL)

Setting WL=1 enables weak locking for the corresponding address region. With WL enabled, all bus cycles are issued with the LOCK# pin negated except for page table accesses and interrupt acknowledge cycles. WL negates bus locking so that previously non-cacheable cycles can be cached. Typically, XCHG instructions, instructions preceded by the LOCK prefix, and descriptor table accesses are locked cycles. Setting WL allows the data for these cycles to be cached.

Weak Locking (WL) implements the same function as NO_LOCK except that NO_LOCK is a global enable. The NO_LOCK bit of CCR1 enables weak locking for the entire address space, whereas the WL bit enables weak locking only for specific address regions.

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.

LBA# Not Asserted (NLB)

Setting NLB=1 prevents the 6x86 processor from asserting the Local Bus Access (LBA#) output pin for accesses to that address region. The RCR regions in combination with the LBA# pin can be used to define local bus address regions. The LBA# signal can then be used by external hardware as an indication that accesses are occurring to the local bus.

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 regardless of the state of KEN#.
- The LBA# pin is asserted.
- Writes are not gathered.
- Strong locking occurs.
- Strong write ordering occurs.
- 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:

- The LBA# pin is asserted.
- Write-back is disabled.
- Writes are not gathered.
- Strong locking occurs.
- Strong write ordering occurs.
- The overlapping regions are non-cacheable.

Example 1: Overlapping Regions with Conflicting Cacheability

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 with conflicting cacheability is a typical configuration for a PC environment. 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 6x86 processor caches accesses as shown in Table 12.

Table 12. Cacheability for Example 1

Address Region	Cacheable	Comments
0 to 640 K bytes	Yes	ARR7/RCE setting.
640 K bytes- 1 M byte	No	NC1 takes precedence over ARR7/RCE setting.
1 M byte - 16 M bytes	Yes	ARR7/RCE setting.
16 M bytes - 4 G bytes	No	Default setting.

Example 2: Overlapping Regions with Conflicting Local Bus Designations

Since the CCR5 bit LBR1 affects LBA# assertion, a potential exists for conflict with the RCR NLB bit, which also affects LBA# assertion. Preferably, regions/bits are defined such that there are no conflicting regions. However, in cases where there is a region overlap the LBR1 bit takes precedence over NLB. For example, for the following settings:

LBR1=1
ARR0 = 0-16 Mbytes
RCR0 NLB=1,

the 6x86 processor LBA# pin behaves as shown in Table 13.

Table 13. LBA# Behavior for Example 2

Address Region	LBA# Behavior	Comments
0 to 640 K bytes	Negated	ARR0/NLB0 setting.
640 K bytes- 1 M byte	Asserted	LBR1 takes precedence over ARR0/NLB0 setting.
1 M byte - 16 M bytes	Negated	ARR0/NLB0 setting.
16 M bytes - 4 G bytes	Asserted	Default setting.

Attributes for Accesses with Conflicting Signal Pin Inputs

The characteristics of the regions defined by the ARRs and the RCRs may also conflict with indications by hardware signals (i.e., KEN#, WB/WT#). The following paragraphs describe how conflicts between register settings and hardware indicators are resolved.

Non-cacheable Regions and KEN#

Regions that have been defined as non-cacheable (RCD=1) by the ARRs and RCRs may conflict with the assertion of the KEN# input. If KEN# is asserted for an access to a region defined as non-cacheable, the access is not cached. Regions defined as non-cacheable by the ARRs and RCRs take precedence over KEN#. The NC1 bit also takes precedence over the KEN# pin. If NC1 is set, any access to the 640K -1M byte address region with KEN# asserted is not cached.

Write-through Regions and WB/WT#

Regions which have been defined as write-through (WT=1) may conflict with the state of the WB/WT# input to the 6x86 processor. Regions defined as write-through by the ARRs and RCRs remain write-through even if WB/WT# is asserted during accesses to these regions. The WT bit in the RCRs takes precedence over the state of the WB/WT# pin in cases of conflict.

Recommended 6x86 Configuration Register Settings

PC Memory Model

Table 14 defines the allowable attributes for a typical PC memory model. Actual recommended configuration register settings for an example PC system are listed in Appendix F.

Table 14. Allowable Attributes for a Typical Memory Model

Address Space	Address Range	Cacheable	Weak Writes	Weak Locks	Write Gathered	Write Through	Notes
DOS Area	0-9FFFFh	Yes	Yes	No	Yes	No	
Video Buffer	A0000-BFFFFh	No	No	No	Yes	No	Note 1
Video ROM	C0000-C7FFFh	Yes	No	No	No	Yes	Note 2
Expansion Card/ROM Area	C8000h-DFFFFh	No	No	No	No	No	
System ROM	E0000h-FFFFFh	Yes	No	No	No	Yes	Note 2
Extended Memory	100000h-Top of Main Memory	Yes	Yes	No	Yes	No	
Unused/PCI MMIO	Top of Main Memory-FFFF FFFFh	No	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). In order for the 6x86 processor to perform 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.

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.

Benchmarking on 6x86 processor systems 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.

3. Top of Main Memory-FFFFFFFFh (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 for 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 32M bytes (0000000-1FFFFFFh) are installed in the system, a non-cacheable region must begin at the 32M byte boundary (2000000h) and extend through the top of the address space (FFFFFFFFh). This is accomplished by using ARR7 (Base = 0000 0000h, BSize=32M bytes) in combination with RCE=1.

General Recommendations

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 "4G byte cache disabled" before executing the memory discovery routine. Once BIOS completes memory discovery, ARR7 should be programmed with a base address of 0000000h 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 1M byte 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 system hangs,
- 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.

I/O Recovery Time (IORT)

Back-to-back I/O writes followed by I/O reads may occur too quickly for a peripheral to respond correctly. Historically, programmers have inserted several "JMP $+2" instructions in the hope that code fetches on the bus would create sufficient recovery time. The 6x86 processor's Branch Target Buffer (BTB) typically eliminates these external code fetches, thus the previous method of guaranteeing I/O recovery no longer applies. For the 6x86 processor, one approach to dealing with this issue is to insert I/O write cycles to a dummy port. I/O write cycles in the form "out imm,reg" are easily implemented as shown below:

OLD IORT	**NEW IORT**
out 21h,al	out 21h,al
jmp $+2	out 80h,al
jmp $+2	out 80h,al
jmp $+2	out 80h,al
in al,21h	in al,21h

The 6x86 processor incorporates an alternative method for implementing I/O recovery time using user-selectable delay settings. See the section on 6x86 processor IORT settings below.

BIOS Creation Utilities

BIOS creation utilities or setup screens must have the capability to easily define and modify the contents of the 6x86 processor configuration registers. This allows OEMs and integrators to easily configure these register settings with the values appropriate for their system design.

Recommended Bit Settings

NC1

The NC1 bit in CCR0 is a predefined non-cacheable region from 640K to 1M byte. The 640K to 1M byte 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.

Suggested setting: NC1 = 1

NO_LOCK

NO_LOCK enables weak locking for the entire address space. NO_LOCK may cause failures for software that requires locked cycles in order to operate correctly.

Suggested setting: NO_LOCK = 0

LOCK_NW

Once set, LOCK_NW prohibits software from changing the NW bit in CR0.

Suggested setting: LOCK_NW = 0

WPR1

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

Suggested setting: WPR1 = 0 unless ROM areas are cached

LINBRST

Linear Burst (LINBRST) allows for an alternate address sequence for burst cycles. The system logic and motherboard design must also support this feature in order for the 6x86 processor to function properly with this bit enabled. Linear Burst provides higher performance than the default "1+4" burst sequence, but should only be enabled if the system is designed to support it.

If the system does support linear burst, BIOS should enable this feature in both the system logic and the 6x86 processor prior to enabling the L1 cache.

Suggested setting: LINBRST = 0 unless linear burst is supported by the system

MAPEN

When set to 1h, the MAPEN bits allow access to all 6x86 processor configuration registers including indexes 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 after the access is complete.

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

IORT

I/O recovery time specifies the minimum number of bus clocks between I/O accesses for the CPU's bus controller. The system logic typically also has a built-in method to select the amount of I/O recovery time. It is preferred to configure the system logic with the I/O recovery time setting and set the CPU for a minimum I/O recovery time delay.

Suggested setting: IORT(2-0) = 7

DTE_EN

DTE_EN allows Directory Table Entries (DTE) to be cached on the 6x86 processor. This provides a performance improvement for some applications that access and modify the page tables frequently.

It has been found that a version of UNIX is not compatible with the DTE caching feature. This version of UNIX incorrectly modifies the page table entries without flushing the TLB. Therefore, it is recommended that the DTE cache remain disabled.

Suggested setting: DTE_EN = 0

CPUIDEN

When set, the CPUIDEN bit enables the CPUID instruction and CPUID detection. By default, the CPUID instruction is disabled (CPUIDEN=0). In the default state, the CPUID opcode 0FA2 causes an invalid opcode exception. Additionally, the CPUID bit in the EFLAGS register cannot be modified by software. 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.

CPUID is disabled by default to guarantee compatibility with popular software that improperly uses CPUID and misidentifies the 6x86 processor. Misidentification of the processor can eventually result in runtime failures.

Suggested setting: CPUIDEN = 0

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 for Windows applications.

Suggested setting: WT_ALLOC = 1

LBR1

LBR1 when set causes the LBA# (Local Bus Access) pin to be asserted for accesses between 640K and 1M byte. This feature is not used for most systems.

Suggested setting: LBR1 = 0

ARREN

The ARREN bit enables/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.

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

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 0000000h and with a "Size" equal to the amount of main memory installed in the system. Memory accesses outside this region are defined as non-cacheable to ensure compatibility with PCI devices.

Suggested setting: ARR7 Base Addr = 0000 0000h
ARR7 Block Size = amount of main memory
RCR7 RCE = 1
RCR7 WWO = 1
RCR7 WL = 0
RCR7 WG = 1
RCR7 WT = 0
RCR7 NLB = 0

SMM Features

The 6x86 processor 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.

USE_SMI

Prior to servicing SMI interrupts, SMM-capable systems must enable the SMM pins by setting USE_SMI=1. The SMM hardware pins (SMI# and SMIACT#) are disabled by default.

SMAC

If set, any access to addresses within the SMM address space accesses SMM memory instead of main memory. Setting SMAC allows access to the SMM memory without servicing an SMI. Additionally, SMAC allows use of the SMINT instruction (software SMI). This bit may be enabled to initialize or test SMM memory but should be cleared for normal operation.

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 be set to 0 to allow for system debug.

Suggested settings for systems not using SMM:

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

Suggested settings for systems using SMM:

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

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 = 0

USE_SUSP

In addition to the HLT instruction, low power suspend mode may be activated using the SUSP# input pin. In response to the SUSP# input, the SUSPA# output indicates when the 6x86 processor has entered low power suspend mode. Systems that support the 6x86 processor's low power suspend feature via the hardware pins must set USE_SUSP to enable these pins.

Suggested setting: USE_SUSP = 0 unless hardware suspend pins supported

Programming Model Differences

Instruction Set

The 6x86 processor supports the 486 instruction set. Pentium extensions for virtual mode, additional debug capability, and internal counters are not supported.

Configuring Internal 6x86 Features

The 6x86 processor supports configuring internal features through I/O ports. The 6x86 processor does not support configuring internal features through the WRMSR and RDMSR instructions which are treated as invalid opcodes.

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 special bus cycles to indicate that external caches should also write back modified data and invalidate their contents.

On the 6x86 processor, the INVD functions identically to the WBINVD instruction. The 6x86 processor always writes all modified internal cache data to external memory prior to invalidating the internal cache contents. In contrast, the Pentium processor invalidates the contents of its internal caches without writing back the "dirty" data to system memory. The Pentium behavior can potentially result in a data incoherency between the CPU's internal cache and system memory.[1]

Control Register 0 (CR0) CD and NW Bits

The CPU's CR0 register contains, among other things, the CD and NW 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. Table 15 lists the cache operating modes for the 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 6x86 processor 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 processor, but is allowed on the 6x86 processor to globally enable write-through caching.[2]

---

[1] See *Pentium Family User's Manual, Volume 3: Architecture and Programming Manual*

[2] Ibid.

Table 15. 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 causse line fills. Write hits update the cache and system emory. 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.

Appendix A - Sample Code: Detecting a IBM CPU

```
assume  cs:_TEXT
public  _isIBM
_TEXT   segment byte public 'CODE'
;*****************************************************************
;       Function: int isIBM ()
;
;       Purpose:        Determine if IBM CPU is present
;       Technique:      IBM CPUs do not change flags where flags
;                       change in an undefined manner on other CPUs
;       Inputs:         None
;       Output:         ax == 1 if IBM 6x86 present, 0 if not
;*****************************************************************
_isIBM          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_IBM           ; flags changed, therefore NOT IBM
                mov   ax, 1             ; TRUE IBM CPU
                jmp   done

not_IBM:
                mov  ax, 0              ; FALSE NON-IBM CPU
done:
                ret
_isIBM          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
     6x86_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 6x86_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 6x86_IDIV_CLKS  17        //known clock counts for 6x86 processor
   #define 6x86_LOOP_CLKS  1
   #define P54_IDIV_CLKS 46          //known clock counts for P54
   #define P54_LOOP_CLKS 7

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

 main(){

 /* declarations */
  unsigned char     uc_IBM_cpu = 0;  //IBM 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 IBM CPU is present ************** */

 //dectect if IBM CPU is present
 uc_IBM_cpu = isIBM();                          //1=IBM, 0=non-IBM

 //display a msg
 if(uc_IBM_cpu) printf("\nIBM CPU present! ");
   else printf("\nIBM 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 6x86 processor or P54
 if(uc_IBM_cpu)ui_idiv=6x86_IDIV_CLKS; else ui_idiv=P54_IDIV_CLKS;
 if(uc_IBM_cpu)ui_loop=6x86_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 6x86 processor 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 value        READ

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

mov    al, 0c0h     ;set index for CCR0
out    22h, al      ;select CCR0 register
mov    al, ah
out    23h,al       ;WRITE new value to CCR0        WRITE
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 value             READ

mov    ah, al
and    ah,0Fh       ;clear upper nibble of ah
or     ah, 2h       ;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 CCR3        WRITE
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=0 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 cacheing
mov     cr0, eax               ;control register 0 write
wbinvd
```

Appendix F - Example Configuration Register Settings

Below is an example of optimized 6x86 processor 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, only WWO and 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.

Register	Bit(s)	Setting	Description
CCR0	NC1	1	Disables caching from 640k-1MByte.
CCR1	USE_SMI SMAC NO_LOCK SM3	1 0 0 1	Enables SMI# and SMIACT# pins. Always clear SMAC for normal operation. Enforces strong locking for compatibility. Sets ARR3 as SMM address region.
CCR2	LOCK_NW SUSP_HLTWP R1 USE_SUSP	0 0 0 0	Locking NW bit not required. Power management not required for this system. ROM areas not cached, so WPR1 not required. Power management not required for this system.
CCR3	SMI_LOCKN MI_EN LINBRST MAPEN(3-0)	1 0 0 0	Locks SMI feature as initialized. Servicing NMIs during SMI not required. Linear burst not supported in this system. Always clear MAPEN for normal operation
CCR4	IORT(2-0) DTE_EN CPUIDEN	7 0 0	Sets IORT to minimum setting. DisablesDTE cache for compatibility. Disables CPUID instruction for compatibility.
CCR5	WT_ALLOC LBR1 ARREN	1 0 1	Enables write allocation for performance.LBA# pinnot required.Enables all ARRs.
ARR0	BASE ADDR BLOCK SIZE	A0000h 7h	Video buffer base address = A0000h. Video buffer block size = 128KBytes.
RCR0	RCD WWO WL WG WTN LB	1 0 0 1 0 0	Caching disabled for compatibility. Caching also disabled via NC1. Write gathering enabled for performance.
ARR1	BASE ADDR BLOCK SIZE	C0000h 8h	Expansion Card/ ROM base address =C0000h. Expansion Card/ROM block size = 256KBytes.
RCR1	RCD WWO WL WG WT NLB	1 0 0 0 0 0	Caching disabled for compatibility. Caching also disabled via NC1.
ARR3	BASE ADDR BLOCK SIZE	A0000h 4h	SMM address region base address SMM address space = 32 KBytes

Register	Bit(s)	Setting	Description
RCR3	RCD WWO WL WG WT NLB	1 0 0 1 0 0	Caching disabled due to overlap with video buffer. Write gathering enabled due to overlap with video buffer.
ARR7	BASE ADDR BLOCK SIZE	0h 7h	Main memory base address = 0h. Main memory size = 16 MBytes.
RCR7	RCE WWO WL WG WT NLB	1 1 0 1 0 0	Caching, weak write ordering, and writegathering enabled for main memory.
ARR(2,4-6)	BASE ADDR BLOCK SIZE	0 0	ARR(2,4-6) disabled (default state).
RCR(2,4-6)	RCD WWO WL WG WT NLB	0 0 0 0 0 0	RCR(2,4-6) not required due to corresponding ARRs disabled (default state).