REVISION DATE	DESCRIPTION OF CHANGES
none	First release
January 22, 1997	Table 1 added summarizing IBM 6x86MX processor and IBM 6x86 processor differences, pg. 4
February 24, 1997	Table 3 was changed, pending product release, pg. 8

May 5, 1997	Added:
	  Time Stamp Counter method for determining operating frequency, pg. 8
	  Configuration Control Register 6 (CCR6), pg. 14
	  Chapter on Model Specific Registers, pg.25-30

May 20, 1997	Changed name to IBM 6x86MX Microprocessor
July 11, 1997	Added two part numbers in 2x clock mode, pg. 8

Introduction

Scope

This document is intended for IBM 6x86MX™ processor[1] system BIOS writers. It is not a stand alone document, but supplements the IBM 6x86MX processor databook. This document highlights the programming differences between the IBM 6x86 processor and the IBM 6x86MX processor. Recommendations for IBM 6x86MX processor 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 others are described in detail.

IBM Configuration Registers

The IBM 6x86MX processor uses on-chip configuration registers to control the on-chip cache, system management mode (SMM), device identification, and other IBM 6x86MX processor-specific features. The on-chip registers are used to activate advanced performance features enhancing. These performance features may be enabled "globally" in some cases, or by a user-defined address region. The flexible configuration of the IBM 6x86MX processor is intended to fit a wide variety of systems.

1. The IBM 6x86MX and 6x86 microprocessors are designed by Cyrix Corp. and manufactured by IBM Microelectronics

The Importance of Non-Cacheable Regions

The IBM 6x86MX 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 IBM 6x86MX 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 IBM 6x86MX processor.

Summary of IBM 6x86MX and 6x86 Processor Differences.

Item	IBM 6x86MX Processor	IBM 6x86 Processor	Page
L1 Cache Size	64 KBytes	16 KBytes	4
DIR0 (Register Index = FEh)	5xh	3xh	5
CPUID (Bit 7 of CCR4)	Reset Value = 1	Reset Value = 0	6, 13
EDX	Reset Value = 06 + DIR0	Reset Value = 05 + DIR0	6
Family Code	06h	05h	7
Time Stamp Counter	Yes	No	8, 25
DTE_EN (Bit 4 of CCR4)	Reserved	If = 1, DTE cache is enabled.	13
SLOP (Bit 1 of CCR5)	Reserved	If =1, the LOOP instruction is slowed down.	13
LBR1 (Bit 4 of CCR5)	Reserved	If =1, LBA# pin is asserted for all accesses to the 640KBytes - 1MByte address region.	13
WWO (Bit 1 of RCRx)	Reserved	If = 1, weak write ordering is enabled for the corresponding region.	16

Table 1: Summary of IBM 6x86MX and IBM 6x86 Processor Differences

Cache Unit

The cache size of the IBM 6x86MX processor was increased to 64 Kbyte. This is 4 times larger than the 16 Kbyte cache of the IBM 6x86 processor. The cache is configured the same way as the IBM 6x86 processor: 4-way set associative, and 32 Byte lines.

IBM 6x86MX CPU Detection

Two of the methods for detecting the IBM 6x86MX CPU are described below. IBM does not recommend other 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.

Detecting the IBM 6x86MX CPU: Method 1

This method for detecting the presence of an IBM 6x86MX 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.

IBM CPU Detection

Detection of an IBM brand CPU is implemented by checking the state of the undefined flags following execution of the divide instruction which divides 5 by 2 (5÷2). The undefined flags in an IBM microprocessor remain unchanged following the divide. Alter-

nate CPUs modify some of the undefined flags. Using operands other than 5 and 2 may prevent the algorithm from working correctly. *Appendix A - Sample Code: Detecting an IBM CPU* contains sample code for detecting an IBM CPU using this method.

Detecting CPU Type and Stepping

Once an IBM brand CPU is detected, the model and stepping 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. Table 2 specifies the contents of the IBM 6x86MX processor DIRs.

DIR0 bits [7:4] = 5h indicate an IBM 6x86MX CPU is present, DIR0 bits [3:0] 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.

Device	CORE/BUS Clock Ratio	DIR0 (Device id)	DIR1 (Rev ID)
IBM 6x86MX Processor	2/1 (default) 2.5/1 3/1 3.5/1	51h or 59h 52h or 5Ah 53h or 5Bh 54h or 5Ch	TBD

Table 2: Device Identification Registers

Detecting the IBM 6x86MX CPU: Method 2

Unlike the IBM 6x86 processor, the CPUID instruction is enabled following reset. It can be disabled by clearing the CPUID bit in configuration register CCR4. It is recommended that all BIOS vendors include a CPUID enable/disable field in the CMOS setup to allow the end-user to disable the CPUID instruction.

The CPUID instruction, opcode 0FA2h, provides information indicating IBM as the vendor and the family, model, stepping, and CPU features. 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..

```
        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]    := 00h
                  EAX[15:8]   := 06h
                  EDX[0]      := 1     /* 1=FPU Built In */
                  EDX[1]      := 0     /* 0=No V86 enhancements */
                  EDX[2]      := 1     /* 1=I/O breakpoints */
                  EDX[3]      := 0     /* 0=No page size extensions */
                  EDX[4]      := 1     /* 1=Time Stamp Counter */
                  EDX[5]      := 1     /* 1=RDMSR and WRMSR */
                  EDX[6]      := 0     /* 0=No physical address extensions */
                  EDX[7]      := 0     /* 0=No machine check exception */
                  EDX[8]      := 1     /* 1=CMPXCHG8B instruction */
                  EDX[9]      := 0     /* 0=No APIC*/
                  EDX[11-10] := 0     /* Undefined *
                  EDX[12]     := 0     /* 0=No memory type range registers */
                  EDX[13]     := 1     /* 1=PTE global bit */
                  EDX[14]     := 0     /* 0=No machine check architecture */
                  EDX[15]     := 1     /* 1=CMOV, FCMOV, FCOMI instructions */
                  EDX[22-16] := 0     /* Undefined */
                  EDX[23]     := 1     /* 1=MMX instructions */
                  EDX[31-24] := 00h  /*                              */

                  break

          default:
                  EAX, EBX, ECX, EDX : Undefined
          }
```

Figure 1. Information Returned by CPUID 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 IBM 6x86MX 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 06xxh. EDX [31-0] will contain the value 0080A135h

EDX Value following Reset

Some CPU detection algorithms may use the value of the CPU's EDX register following reset. The IBM 6x86MX processor's EDX register contains the data shown below following a reset initiated using the RESET pin:

> EDX[31:16]   = undefined
> EDX[15:8]   = 06h
> EDX[7:0]   = DIR0

Refer to Table 2 for DIR0 values. The value in EDX does not identify the vendor of the CPU. Therefore, EDX alone cannot be used to determine if a Cyrix 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.

Determining IBM 6x86MX Processor 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 IBM 6x86MX processor's core is known, DIR0 bits (2:0) can be examined to calculate the bus operating frequency.

Instruction Count Method

Careful selection of instructions and operands must be used to replicate the exact clock counts detailed in the Instruction Set Summary found in the *IBM 6x86MX Microprocessor Databook*. An example code sequence for determining the IBM 6x86MX processor's operating frequency is detailed in Appendix B (assembly language) and Appendix C (C language). This code sequence is identical to the recommended sequence for the IBM 6x86 processor. 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 IBM 6x86MX 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 IBM 6x86MX 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 appendices 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.

Time Stamp Counter Method

On the IBM 6x86MX Microprocessor, the Time Stamp Counter (TSC) can be used as an alternative method for obtaining an exact core clock count during the software timing loop.

The Time Stamp Counter is a 64-bit counter that counts internal CPU clock cycles since the last reset. The value can be read any time via 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. Then the TSD flag is 1, the RDTSC instruction can only be executed at privilege level 0.

The exact core count during the software timing loop can be determined by computing the difference of the Time Stamp Counter at the start of the loop and the end of the loop.

Device Name

The correspondence between core and bus frequency is shown in Table 3. The device name below should be used by the BIOS for display during boot-up and in setup screens or utilities.

Device Name	Frequency (MHz)	
	Core	Bus
IBM 6x86MX Processor 60/150 PR166	150	60
IBM 6x86MX Processor 66/133 PR166	133	66
IBM 6x86MX Processor 66/166 PR200	166	66
IBM 6x86MX Processor 75/150 PR200	150	75
IBM 6x86MX Processor 75/188 PR233	188	75

Table 3: IBM 6x86MX Processor Names

IBM 6x86MX Processor Configuration Register Index

On-chip configuration registers are used to control the on-chip cache, system management mode and other IBM 6x86MX 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 - Sample Code: Programming IBM 6x86MX CPU Configuration Registers* contains example code for accessing the IBM 6x86MX Processor configuration registers.

IBM 6x86MX Processor Configuration Register Index Assignments

Table 4 lists the IBM 6x86MX 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 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. Table 4 lists the MAPEN values required to access each IBM 6x86MX 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
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	Configuration Control 6	CCR6	8	1h
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	x
FFh	Device Identification 1	DIR1	8	x
				x = Don't care

Table 4: Configuration Register Index Assignments

The IBM 6x86MX Processor configuration registers can be grouped into four areas:

1. Configuration Control Registers (CCRs)
2. Address Region Registers (ARRs)
3. Region Control Registers (RCRs)
4. Device Identification Registers (DIRs)

CCR bits independently control IBM 6x86MX Processor features. ARRs and RCRs define regions of memory with specific attributes. DIRs are used for CPU detection as discussed earlier in the last section. 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 the next section.

Configuration Control Registers (CCR0-6)

There are seven CCRs in the IBM 6x86MX 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

Bit7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RSVD	RSVD	RSVD	RSVD	RSVD	RSVD	NC1	RSVD

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.

Table 5: CCR0 Bit Definitions

Configuration Control Register 1 (CCR1)

Bit7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SM3	RSVD	RSVD	NO_LOCK	RSVD	SMAC	USE_SMI	RSVD

Bit Name	Bit No.	Description
SM3	7	If = 1, designates Address Region Register 3 as SMM address space.
NO_LOCK	4	If = 1, 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 non-cacheable 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 = 1, 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 = 1, SMI# and SMIACT# pins are enabled. If = 0, SMI# pin is ignored and SMIACT# pin is driven inactive.

Table 6: CCR1 Bit Definitions

Configuration Control Register 2 (CCR2)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
USE_SUSP	*RSVD*	*RSVD*	WPR1	SUSP_HLT	LOCK_NW	SADS	*RSVD*

Bit Name	Bit No.	Description
USE_SUSP	7	If = 1, SUSP# and SUSPA# pins are enabled. If = 0, 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 = 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 cautiously 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
SADS	1	If = 1, the CPU inserts an idle cycle following sampling of BRDY# and prior to asserting ADS#.

Table 7: CCR2 Bit Definitions

Configuration Control Register 3 (CCR3)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	MAPEN			*RESV*	LINBRST	NMI_EN	SMI_LOCK

Bit Name	Bit No.	Description
MAPEN	7-4	If set to 0001 binary (1h), all configuration registers are accessible. If set to 0000, only configuration registers with indexes C0-CFh, FEh and FFh are accessible.
LINBRST	2	If = 1, the IBM 6x86MX Processor will use a linear address sequence when performing burst cycles. If = 0, the IBM 6x86MX Processor will use a "1+4" address sequence when performing burst cycles. The "1+4" address sequence is compatible with the Pentium's burst address sequence.
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.

Table 8: CCR3 Bit Definitions

Configuration Control Register 4 (CCR4)

The IBM 6x86 processor DTE_EN bit has been eliminated on the IBM 6x86MX Processor, therefore bit 4 of CCR4 is a reserved bit.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CPUID	*RSVD*	*RSVD*	*RSVD*	*RSVD*	IORT		

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.
IORT	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 end of one I/O cycle to the beginning of the next (i.e. BRDY# 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

Table 9: CCR4 Bit Definitions

Configuration Control Register 5 (CCR5)

The 6x86 Slow Loop Instruction (SLOP) and Local Bus Access (LBR1) features have been eliminated in the IBM 6x86MX Processor. Therefore, bits 4 and 1 of CCR5 are a reserved bits on the IBM 6x86MX Processor.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RSVD	*RSVD*	ARREN	*RSVD*	*RSVD*	*RSVD*	*RSVD*	WT_ALLOC

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 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.

Table 10: CCR5 Bit Definitions

Configuration Control Register 6 (CCR6)   Configuration Control Register 6 has been added to the IBM 6x86MX processor.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RSVD	N	*RSVD*	*RSVD*	*RSVD*	*RSVD*	WP_ARR3	SMM_MODE

Bit Name	Bit No.	Description
N	6	Nested SMI Enable bit: If operating in Cyrix enhanced SMM mode and: If = 1: Enables nesting of SMI's If = 0: Disable nesting of SMI's. This bit is automatically CLEARED upon entry to every SMM routine and is SET upon every SMM reoutine and is SET upon every RSM. Therefore, enableing/disabling of nested SMI can only be done while operating in SMM mode.
WP_ARR3	1	If = 1: Memory region defined by ARR3 is write protected when operating outside of SMM mode. If = 0: Disable write protection of memory region define by ARR3. Rest State = 0.
SMM_MODE	0	If = 1: Enables Cyrix Enhanced SMM mode. If = 0: Disables Cyrix Enhanced SMM mode.

Table 11: CCR5 Bit Definitions

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 12 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 13. 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.

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 12: ARRx Index Assignments

BSIZE(3-0)	ARR(0-6) 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

Table 13: BSIZE (3-0) Bit Definitions

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. New to the IBM 6x86MX Processor 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, WL) will be applied to all memory addresses outside the region specified in the corresponding ARR.

If the INV_RGN bit is cleared, the IBM 6x86MX Processor functions identically to the IBM 6x86 processor (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. The IBM 6x86 processor Weak Write Ordering (WWO) and Local Bus Access (NLB) features have been eliminated on the IBM 6x86MX Processor. Therefore, bit 5 and bit 1 are reserved bits for the IBM 6x86MX Processor.

Region Control Registers (RCR0-7

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RSVD	INV_RGN	*RSVD*	WT	WG	WL	*RSVD*	RCD/RCE

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

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.
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.
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.

Table 14: RCR Bit Definitions

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 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 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)

Seting 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.

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#.

- 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:

- Write-back is disabled.

- Writes are not gathered.

- Strong locking occurs.

- 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 IBM 6x86MX Processor caches accesses as shown in Table 15**.**

Attributes for Accesses with Conflicting Signal

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.

Table 15: Cacheability

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 which 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 640 Kbyte -1 MByte 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 IBM 6x86MX 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 IBM 6x86MX Processor Configuration Register Settings

PC Memory Model

Table 16 defines the allowable attributes for a typical PC memory model. Actual recommended configuration register settings for a typical PC system are listed in Appendix F.

Address Space	Address Range	Cacheable	Weak Locks	Write Gathered	Write-through	Notes
DOS Area	0-9FFFFh	Yes	No	Yes	No	
Video Buffer	A0000-BFFFFh	No	No	Yes	No	1
Video ROM	C0000-C7FFFh	Yes	No	No	Yes	2
Expansion Card/ROM Area	C8000h-DFFFFh	No	No	No	No	
System ROM	E0000h-FFFFFh	Yes	No	No	Yes	2
Extended Memory	100000h-Top of Main Memory	Yes	No	Yes	No	
Unused/PCI MMIO	Top of Main Memory-FFFFFFFFh	No	No	No	No	3

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). 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. No benefit to caching these ROM areas on an IBM 6x86MX processor has been seen. 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 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 (0000000-1FFFFFFh) are installed in the system, a non-cacheable region must begin at the 32 MByte boundary (2000000h) and extend through the top of the address space (FFFFFFFFh). This is accomplished by using ARR7 (Base = 0000 0000h, BSize=32MBytes) in combination with RCE=1

Table 16: PC Memory Model

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 "4 GByte 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 1MByte area. Failure to program ARR7 with the correct amount of main memory can result in:

1. Incorrect memory sizing by the operating system eventually resulting in failure,

2. PCI devices not working correctly or causing the system to hang,

3. 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 IBM 6x86MX 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 IBM 6x86MX 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 of "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 IBM 6x86MX processor incorporates an alternative method for implementing I/O recovery time using user selectable delay settings. See the section on IBM 6x86MX 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 IBM 6x86MX 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
Recommended setting: NC1 = 1

The NC1 bit is a 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.

NO_LOCK
Recommended setting: NO_LOCK = 0

The NO_LOCK bit enables weak locking for the entire address space. NO_LOCK may cause fail-ures for software that requires locked cycles in order to operate correctly.

LOCK_NW
Recommended setting: LOCK_NW = 0

Once set, LOCK_NW prohibits software from chang-ing the NW bit in CR0. Since the definition of the NW bit is the same for both the IBM 6x86MX proces-sor and the Pentium processor, it is not necessary to set this bit.

WPR1
Recommended setting: WPR1 = 0 unless ROM areas are cached.

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.

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

Linear Burst allows for an alternate address sequence for burst cycles. The system logic, L2 cache and motherboard design must also support this feature in order for the IBM 6x86MX rocessor 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 IBM 6x86MX rocessor prior to enabling the L1 cache. Appendix H includes sample code that can be used to detect if the L2 cache supports linear burst mode.

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

When set to 1h, the MAPEN bits allow access to all IBM 6x86MX Processor 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.

IORT
Recommended setting: IORT(2-0) = 7

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 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 mini-mum I/O recovery time delay.

CPUID
Recommended setting: CPUID = 1

When set, the CPUID bit enables the CPUID instruc-tion. 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.

WT_ALLOC
Recommended setting:WT_ALLOC = 1

Write Allocate 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 in Windows applications.

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

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.

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 of this region are defined as non-cacheable to ensure compatibility with PCI devices.

Recommended setting:

ARR7 Base Addr   = 0000 0000h
ARR7 Block Size   = amount of main memory
RCR7 RCE   = 1
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 Table 17. All unused/PCI memory space must always be set as non-cacheable.

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						

Table 17: ARR Setting for Various Main Memory Sizes

SMM Features

The IBM 6x86MX 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 are directed to SMM memory instead of main memory. Setting SMAC allows access to the SMM memory without servicing an SMI. Also, 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 = 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
Suggested setting: SUSP_HALT = 0

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.

USE_SUSP
Suggested setting: USE_SUSP = 0 unless hardware suspend pins supported.

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 IBM 6x86MX processor has entered low power suspend mode. Systems that support the IBM 6x86MX processor's low power suspend feature via the hardware pins must set USE_SUSP to enable these pins.

Model Specific Registers

The IBM 6x86MX CPU contains four model specific registers (MSR0 - MSR3). These 64-bit registers are listed in the table below.

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

Table 18: Machine Specific Register

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.

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 IBM 6x86MX processor 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.

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 performance monitor counters use a continuous CPU core clock and will continue to count clock cycles even when the IBM 6x86MX processor is in suspend mode or shutdown.

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.

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 IBM 6x86MX processor. The control register for the two event counters is described in the *IBM 6X86MX Microprocessor Databook*

PM Pin Control

The Counter Event Control register (MSR11) contains PM control fields that define the PM0 and PM1 pins as counter overflow indicators or counter event indicators. When defined as event counters, the PM pins indicate that one or more events occurred during a particular clock cycle and do not count the actual events.

When defined as overflow indicators, the event counters can be preset with a value less the 2^{48}-1 and allowed to increment as events occur. When the counter overflows the PM pin becomes asserted.

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.

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.

26	25	24	23	22	21 16	15 10	9	8	7	6	5 0
TC1*	PM1	CT1	CP13	CP12	TC1*	Reserved	TC0*	PM0	CT0	CP03	CP02	TC0*

Table 19: Counter control Register

Note: Split Fields

Bit Position	Name	Description
25.00	PM1	Define External PM1 Pin If = 1: PM1 pin indicates counter overflows If = 0: PM1 pin indicates counter events
24.00	CT1	Counter #1 Counter Type If = 1: Count clock cycles If = 0: Count events (reset state).
23.00	CP13	Counter #1 CPL 3 Enable If = 1: Enable counting when CPL=3. If = 0: Disable counting when CPL=3. (reset state)
22.00	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
9.00	PM0	Define External PM0 Pin If = 1: PM0 pin indicates counter overflows If = 0: PM0 pin indicates counter events
8.00	CT0	Counter #0 Counter Type If = 1: Count clock cycles If = 0: Count events (reset state).
7.00	CP03	Counter #0 CPL 3 Enable If = 1: Enable counting when CPL=3. If = 0: Disable counting when CPL=3. (reset state)
6.00	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

Table 20: Counter Event Control Register Bit Definitions

Event Type and Description

The events that can be counted by the performance monitoring counters are listed in Table20. 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 inter-rupts, 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.

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 Liens Written Back	Occurrence
07h	yes	yes	External Inquiries	Occurrence
08h	yes	yes	External Inquiries 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
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

Table 21: Event Type Register

Number	Counter 0	Counter 1	Description	Type
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	Saturatins 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	
36.00	yes	no	Reserved	
36.00	no	yes	Reserved	
37.00	yes	no	Return Predicted Incorrectly	Occurrence
37.00	no	yes	Return Predicted (Correctly and Incorrectly)	Occurrence
38.00	yes	no	MMX Instruction Multiply Unit Interlock	Duration
38.00	no	yes	MODV/MOVQ Store Stall Due to Previous Operation	Duration
39.00	yes	no	Returns	Occurrence
39.00	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	

Table 21: Event Type Register

Programming Model Differences

Instruction Set

The IBM 6x86MX processor supports the Pentium Pro processor instruction set plus MMX instructions. Pentium processor extensions for virtual mode are not supported.

Configuring Internal IBM 6x86MX Processor Features

The IBM 6x86MX processor supports configuring internal features through I/O ports.

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 IBM 6x86MX processor, the INVD functions similarly to the WBINVD instruction. The IBM 6x86MX 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. This may result in a data incoherency between the CPU's internal cache and system memory.

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 22 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 IBM 6x86MX 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 IBM 6x86MX processor to globally enable write-through caching.

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.

Table 22: Cache Operating Modes

Appendix A - Sample Code: Detecting an 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 IBM present, 0 if not
;********************************************************************
_isibmproc   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
      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 M2_IDIV_CLKS  17         //known clock counts for IBM 6x86MX CPU
    #define M2_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 ************** */
```

```
  //detect 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 IBM 6x86MX CPU or P54
  if(uc_ibm_cpu)ui_idiv=M2_IDIV_CLKS; else ui_idiv=P54_IDIV_CLKS;
  if(uc_ibm_cpu)ui_loop=M2_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 IBM 6x86MX CPU Configuration Registers

Reading/Writing Configuration Registers

Sample code for setting NC1=1 in CCR0.

```
pushf                   ;save the if flag
cli                     ;disable interrupts
moval, 0c0h             ;set index for CCR0
out22h, al              ;select CCR0 register
inal, 23h               ;READ current CCR0 valueREAD

movah, al
orah, 2h                ;MODIFY, set NC1 bit    MODIFY

moval, 0c0h             ;set index for CCR0
out22h, al              ;select CCR0 register
moval, ah
out23h,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
moval, 0c3h             ;set index for CCR3
out22h, al              ;select CCR3 register
inal, 23h               ;current CCR3 value     READ

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

moval, 0c3h             ;set index for CCR3
out22h, al              ;select CCR3 register
moval, ah
out23h,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.
moveax, 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 IBM 6x86MX™ CPU[1] 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_HLT WPR1 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_LOCK NMI_EN LINBRST MAPEN(3-0)	0 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) CPUIDEN	7 1	Sets IORT to minimum setting. Enables CPUID instruction.
CCR5	WT_ALLOC ARREN	1 1	Enables write allocation for performance. Enables all ARRs.
ARR0	BASE ADDR BLOCK SIZE	A0000h 6h	Video buffer base address = A0000h. Video buffer block size = 128KBytes.
RCR0	RCD WL WG WT INV_RGN	1 0 1 0 0	Caching disabled for compatibility. Caching also disabled via NC1. Write gathering enabled for performance.
ARR1	BASE ADDR BLOCK SIZE	C0000h 7h	Expansion Card/ ROM base address = C0000h. Expansion Card/ROM block size = 256KBytes.
RCR1	RCD WL WG WT INV_RGN	1 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
RCR3	RCD WL WG WT INV_RGN	1 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 WL WG WT	1 0 1 0	Caching, write gathering enabled for main memory.

---

1. The IBM 6x86MX Microprocessor is designed by Cyrix Corp., and manufactured by IBM Microelectronics.

Register	Bit(s)	Setting	Description
ARR(2,4-6)	BASE ADDR BLOCK SIZE	0 0	ARR(2,4-6) disabled (default state).
RCR(2,4-6)	RCD WL WG WT INV_RGN	0 0 0 0 0	RCR(2,4-6) not required due to corresponding ARRs disabled (default state).

Appendix G - Sample Code: Detecting L2 Cache Burst Mode

```
comment~****************************************************
Purpose:    This example program detects if Linear Burst mode is supported.
Method:     There are 3 components (CPU, chipset, SPBSRAM) that must agree on the burst order.  The CPU and
            chipset burst order can be determined by inspecting each devices internal configuration regis-
            ters.  The SPBSRAM devices must be interrogated by a software algorithm (below) to determine if
            "linear burst mode" is enabled/supported correctly.
Algorithm:  If the CPU and chipset are programmed for linear burst mode and a known data pattern exists in
            memory, then the burst mode of the SPBSRAMs can be determined by performing a cache line burst
            and then inspect the data pattern.
Application:  In this example, the SIS5511 chipset is used with an IBM 6x86MX CPU.
Environment:  This program is a REAL mode DOS program to serve as an example.  This example algorithm should
            be ported to BIOS.
            Warnings: For simplicity, this program does not check to see which CPU or chipset is present.
            Nor, does this program check to see if the CPU is in REAL mode before executing protected
            instructions. Also, this program blindly overwrites data in the 8000h segment of memory.
*********************************************************~
;version m510                    ;remove comment for TASM


DOSSEG
.MODEL   SMALL
.DATA
Msg_1        db     0dh,0ah
             db     'ISLINBUR.EXE checks if L2 SRAMs are in Linear Burst Mode or'
             db     0dh,0ah
             db     'Toggle Burst mode for the SIS5511 chipset and the IBM
                       6x86MX CPU CPU.'
             db     0dh,0ah
             db     '$'
Msg_2        db     0dh,0ah
             db     'Test complete!'
             db     0dh,0ah
             db     '$'
Msg_yes  db    0dh,0ah
             db     'The L2 SRAMs correctly operate in linear burst mode.'
             db      0dh,0ah
             db      '$'
Msg_no    db     0dh,0ah
             db     'ERROR: The L2 SRAMs incorrectly operate in linear burst mode.'
             db      0dh,0ah
             db      '$'


index_port      dw      0CF8h
data_port       dw      0CFCh
pci_index       dd      80000000h


.STACK   100h
```

```
.CODE
.STARTUP
.486P
pushf
cli
;-------display a msg using a DOS call
mov     ax,seg Msg_1
mov     ds,ax
mov     dx,offset Msg_1     ;set msg_1 start
mov     ah,9h               ;print string function
int     21h                 ;DOS int

;-------disable the L1 internal cache
call    cache_off
out     80h,al                       ;write to PC diagnostic port

;-------setup a work space in main memory to perform burst
;mode tests and initialize the memory work space with a
;known pattern
push    ds
mov     ax,8000h            ;choose segment 8000h
mov     ds,ax
mov     al,0001h
mov     byte ptr ds:[0],al    ;init memory locations
inc     al
mov     byte ptr ds:[8h],al
inc     al
mov     byte ptr ds:[10h],al
inc     al
mov     byte ptr ds:[18h],al
pop     ds

;-------enable the SiS5511 chipset's linear burst mode
mov     al,51h              ;al=reg to read
call    r_pci_reg           ;READ al=reg contents
mov     ah,al
or      ah,8                ;MODIFY set linbrst bit
mov     al,51h
call    w_pci_reg           ;WRITE

;-------enable the CPU's linear burst mode
call    en_linbrst

;-------enable L1 caching
call    cache_on

;-------burst several cache lines so that address 80000h is
;in the L2 cache, but NOT in the L1 cache.
```

```
push    ds
mov     ax,8000h              ;choose segment 8000h
mov     ds,ax
mov     al,byte ptr ds:[0h]   ;line fill to L2 and L1
mov     al,byte ptr ds:[1000h]   ;fill L1 line 1
mov     al,byte ptr ds:[2000h]   ;fill L1 line 1
mov     al,byte ptr ds:[3000h]   ;fill L1 line 1
mov     al,byte ptr ds:[4000h]   ;fill L1 line 1,
                        ;now 80000h exists only in the
                        ;L2 cache (not in L1 anymore!)


;-------burst a cache line so that address 80000h will hit
; the L2 cache SRAMs
mov     al,byte ptr ds:[8h]
            ;***** Burst Pattern Table *****
            ;if SRAMs in linear burst mode, then
            ;L1 will be filled with:
                        ; byte data
                        ;  0    01h
                        ;  8    02h
                        ; 10    03h
                        ; 18    04h
            ;if SRAMs in toggle burst mode, then
                    ;L1 will be filled with:
                        ; byte  data
                        ;  0    03h
                        ;  8    02h
                        ; 10    01h
                        ; 18    04h


;-------Compare the cache line to the Burst Pattern Table
;above. The signature of the pattern will determine
;if the burst was linear or toggle.


                                    ;check byte ds:[10] in the L1
mov     al, byte ptr ds:[10h]
                                    ;it will be a 1 if toggle mode
cmp     al,3h
                                    ;it will be a 3 if linear mode
pop     ds
jnz     not_linear

is_linear:
mov     dx,offset Msg_yes ;SRAMs in linear burst mode
jmp     over_not
not_linear:
mov     dx,offset Msg_no  ;SRAMs in toggle burst mode
over_not:
```

```
wbinvd

;------disable L1 internal cache
call    cache_off

;-------restore chipset to toggle mode burst order
mov     al,51h             ;al=reg to read
call    r_pci_reg          ;READ al=reg contents
mov     ah,al
and     ah,0f7h            ;MODIFY clr linbrst bit
mov     al,51h
call    w_pci_reg          ;WRITE

call    dis_linbrst

;-------restore L1 caching
call    cache_on

done:
popf

;-------display a msg using a DOS call
mov     ax,seg Msg_2
mov     ds,ax
mov     ah,9h                   ;print string function
int     21h                     ;DOS int

;--------return to the operating system
.EXIT

comment~*************************************************
 function        r_pci_reg
 purpose         read the pci register at the index in al
 inputs          al= the index of the pci register
 returns         al= the data read from the pci reg
 *********************************************************~
r_pci_reg PROC

        pushf
        push    eax
        push    dx
        cli

        mov     dx,index_port
        and     eax,0FFh
        or      eax,pci_index
        out     dx,eax
```

```
        and     al,3
        mov     dx,data_port
        add     dl,al
        in      al,dx
        xchg    al,bl               ;preserve rtn value


        mov     eax,pci_index
        mov     dx,index_port
        out     dx,eax


        pop     dx
        pop     eax
        popf


        xchg    al,bl
        ret


r_pci_reg ENDP

comment~*************************************************
 function       w_pci_reg
 inputs         al= the index of the pci register
        ah= the data to write
 outputs        modifies chipset registers directly
 returns        none
**********************************************************~
w_pci_reg proc

        pushf
        push    eax
        push    bx
        push    dx
        cli


        mov     bx,ax           ;preserve input value(s)


        mov     dx,index_port
        and     eax,0FFh
        or      eax,pci_index
        out     dx,eax


        and     al,3
        mov     dx,data_port
        add     dl,al
        mov     al,bh               ;recall data to write
        out     dx,al


        mov     eax,pci_index
```

```
        mov       dx,index_port
        out       dx,eax

        pop       dx
        pop       bx
        pop       eax
        popf
        ret
w_pci_reg ENDP


comment~****************************************************
 function        en_linbrst
 purpose         enable the IBM 6x86MX CPU linbrst bit
 inputs          none
 outputs         modifies the IBM 6x86MX CPU registers directly
 returns         none
********************************************************~
en_linbrst PROC

mov     ax,0C3C3h                   ;set LINBRST
out     22h,al
in      al,23h
xchg    ah,al
or      ah,4
out     22h,al
xchg    ah,al
out     23h,al


ret
en_linbrst ENDP


comment~****************************************************
 function        dis_linbrst
 purpose         disable the IBM 6x86MX linbrst bit
 inputs          none
 outputs         modifies the IBM 6x86MX CPU registers directly
 returns         none
********************************************************~
dis_linbrst PROC

mov     ax,0C3C3h
out     22h,al
in      al,23h
xchg    ah,al
and     ah,0fbh        ;clear the linbrst bit
out     22h,al
xchg    ah,al
out     23h,al
```

```
ret
dis_linbrst ENDP

comment~**************************************************
 function        cache_off
 purpose         disables the L1 cache
 inputs          none
 returns         none
************************************************************~
cache_off PROC
pushf
push    eax
cli
mov     eax,cr0
or      eax,60000000h
mov     cr0,eax
wbinvd
jmp     $+2
pop     eax
popf
ret
cache_off ENDP

comment~**************************************************
 function        cache_on
 purpose         enables the L1 cache
 inputs          none
 returns         none
************************************************************~
cache_on PROC
pushf
push    eax
cli
mov     eax,cr0
and     eax,9FFFFFFFh
mov     cr0,eax
pop     eax
popf
ret

cache_on ENDP
END
```

IBM ®

For more information contact your IBM Microelectronics Sales Representative

Or visit our website at http://www.chips.ibm.com