Appendixes for the IBM 6x86MX Microprocessor BIOS Writer's Guide

Application Note

Revision Summary: Please see Document #40250 for the body portion of the IBM 6x86MX Microprocessor BIOS Writer's Guide.

X86 applications notes and databooks can be found on the website at
http://www.chips.ibm.com/products/x86

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
;*****************************************************************
_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
     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
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, 10h           ;MODIFY, set MAPEN(3-0)       MODIFY

mov   al, 0c3h          ;set index for CCR3
out   22h, al           ;select CCR3 register
mov   al, ah
out   23h,al            ;WRITE new value to 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=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	1	Enables SMI# and SMIACT# pins.
	SMAC	0	Always clear SMAC for normal operation.
	NO_LOCK	0	Enforces strong locking for compatibility.
	SM3	1	Sets ARR3 as SMM address region.
CCR2	LOCK_NW	0	Locking NW bit not required.
	SUSP_HLT	0	Power management not required for this system.
	WPR1	0	ROM areas not cached, so WPR1 not required.
	USE_SUSP	0	Power management not required for this system.
CCR3	SMI_LOCK	0	Locks SMI feature as initialized.
	NMI_EN	0	Servicing NMIs during SMI not required.
	LINBRST	0	Linear burst not supported in this system.
	MAPEN(3-0)	0	Always clear MAPEN for normal operation.
CCR4	IORT(2-0)	7	Sets IORT to minimum setting.
	CPUIDEN	1	Enables CPUID instruction.
CCR5	WT_ALLOC	1	Enables write allocation for performance.
	ARREN	1	Enables all ARRs.
ARR0	BASE ADDR	A0000h	Video buffer base address = A0000h.
	BLOCK SIZE	6h	Video buffer block size = 128KBytes.
RCR0	RCD	1	Caching disabled for compatibility. Caching also
	WL	0	disabled via NC1.
	WG	1	
	WT	0	Write gathering enabled for performance.
	INV_RGN	0	
ARR1	BASE ADDR	C0000h	Expansion Card/ ROM base address = C0000h.
	BLOCK SIZE	7h	Expansion Card/ROM block size = 256KBytes.
RCR1	RCD	1	Caching disabled for compatibility. Caching also
	WL	0	disabled via NC1.
	WG	0	
	WT	0	
	INV_RGN	0	
ARR3	BASE ADDR	A0000h	SMM address region base address
	BLOCK SIZE	4h	SMM address space = 32 KBytes
RCR3	RCD	1	Caching disabled due to overlap with video
	WL	0	buffer.
	WG	1	
	WT	0	

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

Register	Bit(s)	Setting	Description
	INV_RGN	0	Write gathering enabled due to overlap with video buffer.
ARR7 RCR7	BASE ADDR BLOCK SIZE RCE WL WG WT	0h 7h 1 0 1 0	Main memory base address = 0h. Main memory size = 16 MBytes. Caching, write gathering enabled for main memory.
ARR(2,4-6) RCR(2,4-6)	BASE ADDR BLOCK SIZE RCD WL WG WT INV_RGN	0 0 0 0 0 0 0	ARR(2,4-6) disabled (default state). 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 deter-
            mined by inspecting each devices internal configuration registers.
            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
```