

Cyrix M2 CPU Software Identification

Version 0.1

1.0 What's new with the M2?

The M2 will have CPUID enabled. Please use CPUID to determine if the M2 is present. Future Cyrix CPUs will have CPUID enabled. EFLAGS bit 21 will be read/writable to indicate support for the CPUID instruction.

The M2 CPUID information is:

Vendor ID String	"CyrixInstead"
CPUID Levels Supported	1
Family	6
Model	0
Stepping	TBD
Feature Flag Value	0x0080A135

Feature Flags Set:

- FPU Present
 - M2 contains an enhanced FPU.
- I/O Breakpoints
 - I/O cycles can be trapped, controllable by CR4:DE.
- Time Stamp Counter Supported
 - RDTSC instruction is supported, controllable by CR4:TSD.
- RDMSR and WRMSR Instructions Present
 - Model Specific Registers are supported.
- CMPXCGH8B Instruction Supported
 - Compare exchange eight byte instruction supported.
- PTE Global Bit Support
 - When set in PTE TLB will not be flushed when CR3 is written.
- CMOV and FCMOV* Instructions Supported
 - Conditional move instructions supported.
- MMX Instructions Supported
 - Multi Media Instruction Extensions supported.

For update to this document, software or complete Cyrix device detection please visit the Cyrix Software Developer WEB page at:

<http://www.cyrix.com/process/support/isv.htm>

2.0 CPUID Technique

The first test to do when detecting Cyrix CPUs is to test for the presence of CPUID. This is advised rather than using the traditional CYRIX 5/2 Flag test since it does not yield information as quickly and completely as CPUID. The traditional Cyrix 5/2 Flag test will still work for the M2.

2.1 Overview

Check for CPUID instruction support

- This is typically done by testing if the ID bit (bit 21) of the EFLAGS Register is read/writable.

If the CPUID instruction is supported:

Read CPUID Levels available

- This will inform software of the number of levels or EAX values that CPUID can be executed with and provide CPU information.

Read CPU Manufacturer or Vender String

- The CPU Manufacture moniker is embedded in the CPUID and can be read by software.

If CPUID level 1 supported:

Read CPU Family, Model, Stepping information

- CPU vital information can be obtained for classification purposes.

Read CPU Feature Support

- CPU facility support can be determined by inspecting the feature information.

2.2 C Code Example

```

if ( is_cpuid_supported () ) /* Check for CPUID instruction support */
{
    /*** execute CPUID with EAX == 0 ***/
    get_cpuid_info (0, &reg_eax, &reg_ebx, &reg_ecx, &reg_edx);

    /*** register EAX now contains the levels of CPUID support ***/
    cpuid_levels = reg_eax;

    /*** build vender id string ***/
    vender[0] = reg_ebx;
    vender[1] = reg_edx;
    vender[2] = reg_ecx;
    vender[3] = 0;

    if (cpuid_levels > 0) /* Check for CPUID level 1 support */
    {
        /*** execute CPUID with EAX == 0 ***/
        get_cpuid_info (1, &reg_eax, &reg_ebx, &reg_ecx, &reg_edx);

        family   = (reg_eax & 0xf00) >> 8;      /* get family   */
        model    = (reg_eax & 0xf0) >> 4;       /* get model    */
        stepping = reg_eax & 0xf;              /* get stepping */

        /*** Check for Cyrix CPU presence ***/
        if ((strcmp ("CyrixInstead", vender) == 0))
        {
            if (family == 5)                    /*** Cyrix 6x86 CPU ***/
                cpu_is = Cyrix_6x86_CPU;

            if (family == 6)                    /*** Cyrix M2 CPU   ***/
                cpu_is = Cyrix_M2_CPU;
        }

        cpu_features = reg_edx;

        else /*** Other CPU Vender CPU Present ***/
            /*** Tests for other CPU Vender CPUs ***/
        } /* cpuid_levels > 0*/
    } /* is CPUID supported */

    /*******
    /*** Code here to determine CPU manufacturer and type for CPUs ***/
    /*** not supporting the CPUID instruction. ***/
    /*******

```

3.0 Source Code

```

assume cs:_TEXT

public _is_cpuid_supported
public _get_cpuid_info

_TEXT segment byte public 'CODE'

;*****
;      Function: int is_cpuid_supported ()
;
;      Purpose:      Check for existence of CPUID instruction.
;                    If exists execute CPUID with eax==0.
;
;      Inputs:       none
;
;      Output:
;
;      Returns:      0 - no CPUID instruction supported
;                    1 - CPUID instruction supported
;*****

cpuid macro
    db 0fh, 0a2h
endm

_is_cpuid_supported    proc    near
.386
    push    bp
    mov bp, sp
    sub    sp, 40

    push    eax
    push    ebx

    pushfd                ; get extended flags
    pop     eax
    mov     ebx, eax      ; save current flags

    xor     eax, 200000h   ; toggle bit 21
    push   eax            ; put new flags on stack
    popfd                ; flags updated now in flags

    pushfd                ; get extended flags
    pop     eax
    xor     eax, ebx      ; if bit 21 r/w then eax <> 0

    pop     ebx
    pop     eax

    je     no_cpuid      ; can't toggle id bit (21) no cpuid here
    mov    ax, 1         ; cpuid supported

    jmp    done_cpuid_sup

```

```

no_cpuid:
    mov     ax, 0                ; cpuid not supported

done_cpuid_sup:
    mov     sp, bp
    pop     bp
    ret

_is_cpuid_supported      endp

;*****
;       Function: int cpuid_supported (long level,
;                                   long &reg_eax,
;                                   long &reg_ebx,
;                                   long &reg_ecx,
;                                   long &reg_edx)
;
;       Purpose:      Execute CPUID instruction at level (eax==level)
;
;       Inputs:       level - eax value when CPU is executed
;
;       Output:       reg_eax - eax after executing CPUID
;                   reg_ebx - ebx after executing CPUID
;                   reg_ecx - ecx after executing CPUID
;                   reg_edx - edx after executing CPUID
;*****

_get_cpuid_info      proc   near
.386
    push   bp
    mov    bp, sp
    sub    sp, 4

    push   esi

    mov    eax, dword ptr [bp+4]    ; get level

    cpuid

    mov    esi, ebx                ; save ebx value

    mov    bx, word ptr [bp+8]
    mov    dword ptr [bx], eax     ; reg_eax := eax

    mov    bx, word ptr [bp+10]
    mov    dword ptr [bx], esi     ; reg_ebx := ebx (via esi)

    mov    bx, word ptr [bp+12]
    mov    dword ptr [bx], ecx     ; reg_ecx := ecx

    mov    bx, word ptr [bp+14]
    mov    dword ptr [bx], edx     ; reg_edx := edx

    pop    esi

    mov    sp, bp
    pop    bp
    ret
_get_cpuid_info      endp

_TEXT ends
end

```