intel®

AP-485

APPLICATION NOTE

Intel Processor Identification with the CPUID Instruction

November 1993

PRELIMINARY

Order Number: 241618-001

Intel Processor Identification with the CPUID Instruction

CONTENTS

CONTENTS

1.0 BACKGROUND

As new generations and new models of processors have been added to the Intel X86 architecture (8086, 8088, Intel 286, Intel386™, Intel486™, and Pentium™ processors), Intel has provided increasingly sophisticated methods to software for identifying the features available on the processor.

- First, Intel started publishing code sequences that identify processor generation by detecting minor differences of implementation.
- Later, with the advent of the Intel386 processor, Intel started providing the CPU signature (family, model, and stepping numbers) to software at reset.
- Ultimately, Intel has extended the X86 architecture with the CPUID instruction. The CPUID instruction not only provides the CPU signature but also provides information about the features supported by the processor.

This latest step is necessary because the computing market is demanding processor models within a given processor generation that have differing sets of features. Anticipating that this trend will continue with future processor generations, Intel has made the CPUID instruction extensible.

The purpose of this Application Note is to show how to use the CPUID instruction in such a way that software can execute compatibly on the widest possible range of Intel X86 generations and models, past, present, and future.

2.0 DETECTING THE CPUID INSTRUCTION

Intel has provided a straightforward method for detecting whether the CPUID instruction is available. This method uses the ID flag in bit 21 of the EFLAGS register. If software can change the value of this flag, the CPUID instruction is available. The program example in Section 5.0 shows how to use the PUSHFD instruction to read and the POPFD instruction to change the value of the ID flag.

3.0 OUTPUTS OF THE CPUID INSTRUCTION

Figure 1 summarizes the outputs of CPUID.

The CPUID instruction can be executed multiple times, each time with a different parameter in the EAX register. The outputs depend on the value of EAX, as Table 1 specifies. To determine the highest acceptable value of the EAX parameter, the program should set EAX to zero. In this case, CPUID returns to EAX the value of the highest parameter that it recognizes. No execution of CPUID should use a parameter greater than this highest value. Although this highest value is 1 for the first model of the Pentium processor, it might be different for future processor models.

3.1 Vendor-ID String

The vendor identification string is stored in the registers EBX, EDX, and ECX in such a way that, when the contents of these registers are stored in memory in adjacent locations (EBX at the lowest address, ECX at the highest), the byte string "GenuineIntel" appears in memory.

While any imitator of the X86 architecture can provide the CPUID instruction, no imitator can legitimately claim that its part is a genuine Intel part. Therefore, the presence of the "GenuineIntel" string is an assurance that the CPUID instruction and the CPU signature are implemented as described in this document.

3.2 CPU Signature

Beginning with the Intel386 processor, the CPU signature has been available at reset. With processors that implement the CPUID instruction, the CPU signature is made available both by reset and by the CPUID instruction. Figure 1 shows the format of the signature on the Intel486 processor, Pentium processor, and later processor generations; Table 2 shows the values that are currently defined. (The high-order 20 bits are undefined and reserved.)

On Intel386 processors, the format of the CPU signature is somewhat different, as Figure 2 shows. Table 3 gives the currently possible values.

Regardless of signature format, the MODEL and FAMILY fields are significant only in processors that do not implement the CPUID instruction.

The STEPPING fields help software deal with errata. Intel releases information about errata and related stepping numbers as needed.

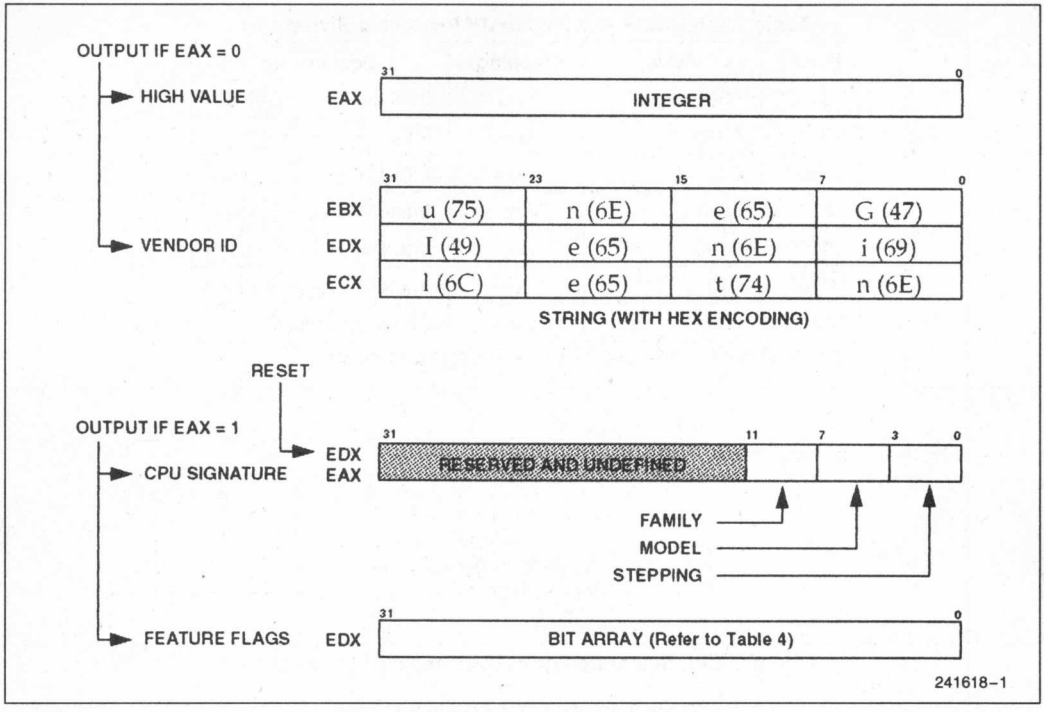

Figure 1. CPUID Instruction Outputs

Table 1. Effects of EAX Contents on CPUID

Parameter	Outputs of CPUID
EAX = 0	EAX ← highest value
	EBX:EDX:ECX ← Vendor Identification String
EAX = 1	EAX ← CPU signature
	EDX ← Feature flags
1 < EAX ≤ highest value	Might be defined in future processor models
EAX > highest value	Undefined

Table 2. Intel486™ and Pentium™ Processor Signatures

Family	Model	Stepping[a]	Description
0100	0000 and 0001	xxxx	Intel486 DX Processor
0100	0010	xxxx	Intel486 SX Processor
0100	0011	xxxx	Intel487™ Processor
0100	0011	xxxx	Intel486 DX2 Processor
0100	0100	xxxx	Intel486 SL Processor
0100	0101	xxxx	Intel486 SX2 Processor
0101	0001	xxxx	Pentium Processor

[a]Intel releases information about stepping numbers as needed.

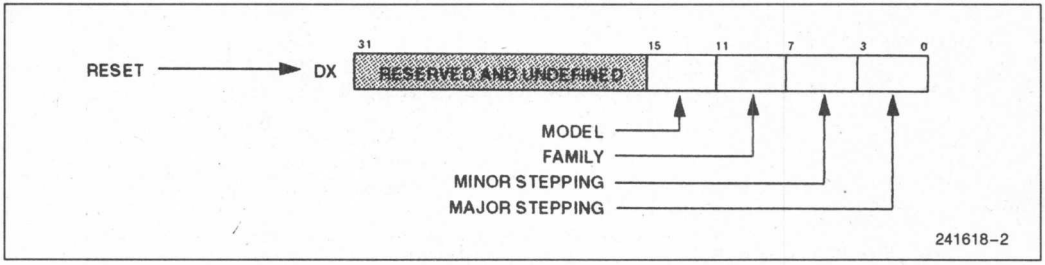

Figure 2. CPU Signature Format on Intel386™ Processors

Table 3. Intel386™ CPU Signatures

Model	Family	Major Stepping	Minor Stepping[a]	Description
0000	0011	0000	xxxx	Intel386 DX Processor
0010	0011	0000	xxxx	Intel386 SX Processor
0011	0011	0000	xxxx	Intel 376 Processor
0100	0011	0000	xxxx	Intel386 SL Processor
0100	0011	0001	xxxx	Intel386 SL Processor (B-step)
0000	0011	0100	xxxx	RapidCAD™ Processor

[a]Intel releases information about minor stepping numbers as needed.

3.3 Feature Flags

The CPUID instruction sets the feature register, EDX, to indicate which features the processor supports. Each set feature flag can indicate either that a feature is present or that a feature is not present. Refer to Table 4 for the meanings of the feature flags that are currently defined. For each future processor generation or model, refer to the programmer's reference manual, user's manual, or equivalent documentation for additional definitions.

Software executing on a processor that supports the CPUID instruction should use the feature flags instead of the model and family to determine what features are present. Doing so helps make software immune to incompatibilities that might arise from the release of an additional model that has a feature set different than that of the model for which the software was designed.

Table 4. Feature Flag Values

Bit Position	Abbreviation	Meaning When Flag = 1
0	FPU	Floating-point unit on-chip
1–6[a]	(see note)	(see note)
7	MCE	Machine-check exception present
8	CX8	CMPXCHG8B instruction present
9–31[a]	(see note)	(see note)

[a]Some non-essential information regarding the Pentium processor is considered Intel confidential and proprietary and has not been documented in this publication. This information is provided in the *Supplement to the Pentium™ Processor User's Manual* and is available with the appropriate non-disclosure agreements in place. Contact Intel Corporation for details.

4.0 USAGE GUIDELINES

This document presents straightforward -feature-detection methods. Software should not try to identify features by exploiting programming tricks, "undocumented features," or otherwise deviating from the intent of this document. The following list gives some tips that can help programmers maintain the widest range of compatibility for their software.

- Do not depend on the absence of an invalid opcode trap on the CPUID opcode to detect CPUID. Do not depend on the absence of an invalid opcode trap on the PUSHFD opcode to detect a 32-bit processor. Test the ID flag, as described in Section 2.0 and shown in Section 5.0 .
- Do not assume that a given family or model has any specific feature. For example, do not assume that, because FAMILY = 5 (-Pentium processor), there must be a floating-point unit on-chip. Use the feature flags for this determination.
- Do not assume that the existence of the CPUID instruction implies a Pentium processor or later generation CPU. Future versions of the Intel486 microprocessor may also include the CPUID instruction.
- Do not use undocumented features of a CPU to identify steppings or features. For example, the Intel386 CPU A-step had bit instructions that were withdrawn with B-step. Some software attempted to execute these instructions and depended on the invalid-opcode exception as a signal that it was not running on the A-step part. This software failed to work correctly when the Intel486 CPU used the same opcodes for different instructions. That software should have used the stepping information in the CPU signature.
- Do not assume that a value of 1 in a feature flag indicates that a given feature is present, even though that is the case in the first model of the Pentium processor. For some feature flags that might be defined in the future, a value of 1 can indicate that the corresponding feature is not present.
- Programmers should be careful to test feature flags individually and not make assumptions about irrelevant bits. It would be a mistake, for example, to test the FPU bit by comparing the feature register to 1 with a compare instruction.
- Do not assume that the clock of a given family or model runs at a specific speed. In particular, do not write clock-dependent code, such as timing loops. An upgrade processor can run at a faster speed, while reporting the same family and model. For an example, refer to the Intel487 and Intel486DX processors in Table 2. These processors have different clock speeds, even though they have the same family and model numbers. Use the systems timers for measuring time.

5.0 PROPER IDENTIFICATION SEQUENCE

The following program example concludes this document by showing correct usage of the CPUID instruction. It also shows how to identify earlier processor generations that implement neither the CPU signature nor the CPUID instruction. This program contains three procedures:

1. **get_cpuid** which identifies the type of CPU. Figure 3 shows the flow of this procedure.
2. **check_fpu** which determines what type of floating-point unit (FPU) or math coprocessor (MCP) is present.
3. **print** which uses DOS function calls to display the results of the other procedures on the monitor of a PC. This procedure can be omitted or modified to execute with other operating systems.

This procedure has been tested with 8086, 80286, Intel386, Intel486, and Pentium processors.

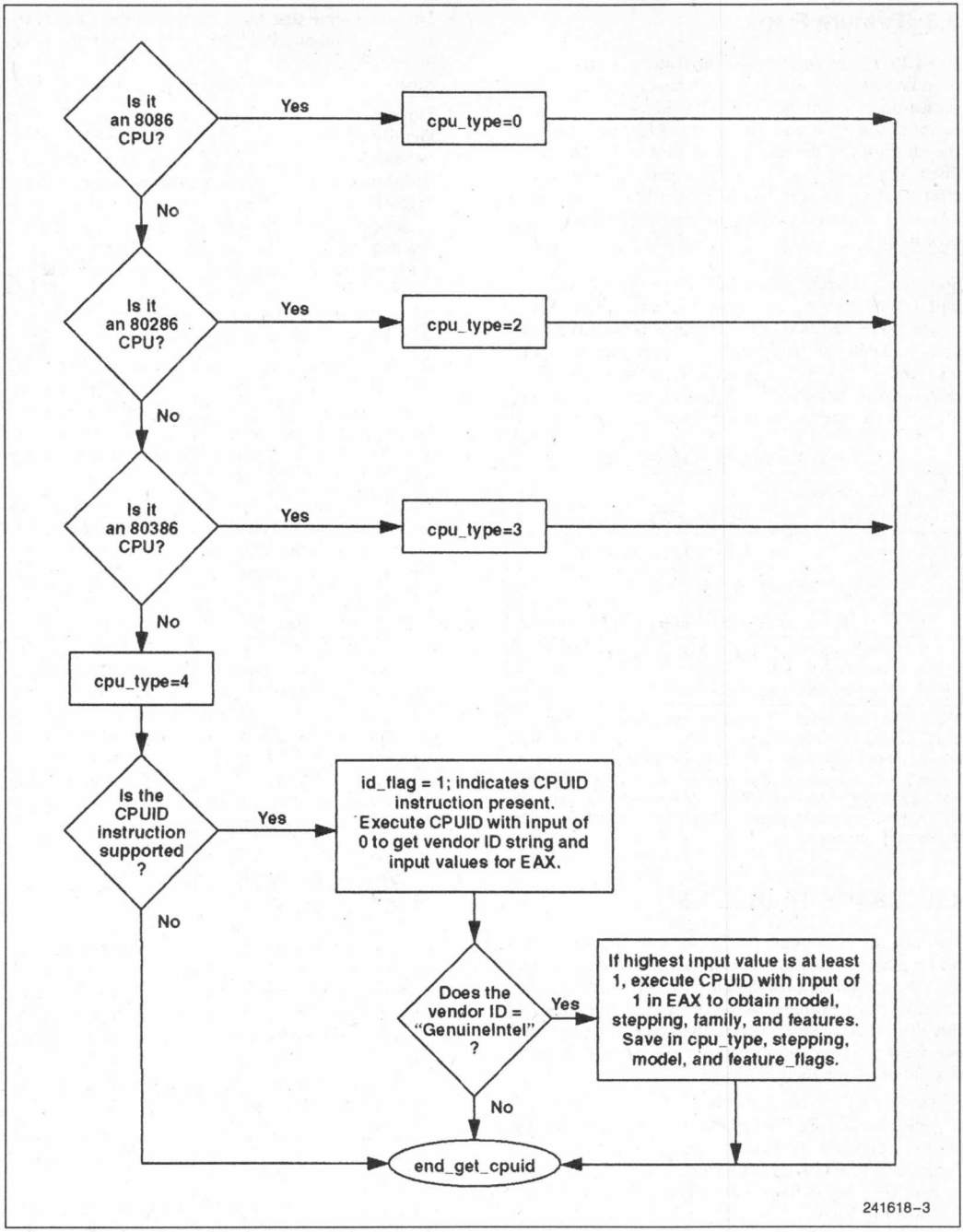

Figure 3. Flow of GET_CPUID Procedure

```
;       Filename:       cpuid32.msm
;
;       This program has been developed by Intel Corporation. You have
;       Intel's permission to incorporate this source code into your
;       product royalty free.
;
;       Intel specifically disclaims all warranties, express or implied,
;       and all liability, including consequential and other indirect
;       damages, for the use of this code, including liability for
;       infringement of any proprietary rights. Intel does not assume
;       any responsibility for any errors which may appear in this code
;       nor any responsibility to update it.
;
;       This program contains three parts:
;       Part 1: Identifies CPU type in the variable cpu_type:
;               0=8086 processor
;               2=Intel 286 processor
;               3=Intel386(TM) processor
;               4=Intel486(TM) processor
;               5=Pentium(TM) processor
;
;       Part 2: Identifies FPU type in the variable fpu_type:
;               0=FPU not present
;               1=FPU present
;               2=287 present (only if cpu_type=3)
;               3=387 present (only if cpu_type=3)
;
;       Part 3: Prints out the appropriate message. This part can
;               be removed if this program is not used in a DOS-based
;               system. Portions affected are at the end of the
;               data segment and the print procedure in the code
;               segment.
;
;       This program was assembled with Microsoft's Assembler MASM 6.0.
;       While this program mostly uses 16-bit operands, some 32-bit
;       operands are required to check the 32-bit EFLAGS register once
;       it has been determined that the processor is a least an
;       Intel386 processor. 32-bit operations are invoked by using
;       the macro OPND32.

        TITLE   CPUID
        DOSSEG
        .model  small
        .stack  100h
        .186

; The OPND32 macro takes either zero or two parameters.
; With zero parameters, it generates the 32-bit operand-size prefix.
; With two parameters, it generates the 32-bit operand-size prefix,
; followed by an opcode and a 32-bit immediate value. These parameters
; are used to generate XOR AX,imm32 instructions.
```

Example 1. CPU Identification Procedure

```
OPND32 MACRO op_code, op_erand
        db      66h       ; Force 32-bit operand size
  IFNB <op_code>
        db      op_code   ; Optional opcode
    IFNB <op_erand>
        dd      op_erand  ; Optional 32-bit immediate value
    ENDIF
  ENDIF
ENDM

CPUID MACRO
        db      0fh       ; Opcode for CPUID instruction
        db      0a2h
ENDM

TRUE            equ     1
FAMILY_MASK     equ     0f00h
FAMILY_SHIFT    equ     8
MODEL_MASK      equ     0f0h
MODEL_SHIFT     equ     4
STEPPING_MASK   equ     0fh
FPU_FLAG        equ     1h
MCE_FLAG        equ     80h
CMPXCHG8B_FLAG  equ     100h

        .data
fp_status       dw      ?
vendor_id       db      12 dup (?)
cpu_type        db      ?
model           db      ?
stepping        db      ?
id_flag         db      0
fpu_type        db      0
intel_proc      db      0
feature_flags   dw      2 dup (0)
;
; Remove the remaining data declarations if not using the DOS-based
; print procedure
;

id_msg          db "This system has a$"
fp_8087         db " and an 8087 math coprocessor$"
fp_80287        db " and an 80287 math coprocessor$"
fp_80387        db " and an 80387 math coprocessor$"
c8086           db "n 8086/8088 processor$"
c286            db "n 80286 processor$"
c386            db "n 80386 processor$"
c486            db "n 80486 DX processor or 80487 SX math coprocessor$"
c486nfp         db "n 80486 SX processor$"
Intel486_msg    db 13,10,"This system contains a Genuine "
                db "Intel486(TM) processor",13,10,"$"
Pentium_msg     db 13,10,"This system contains a Genuine "
                db "Intel Pentium(TM) processor",13,10,"$"
modelmsg        db "Model:              $"
steppingmsg     db "Stepping:           $"
```

Example 1. CPU Identification Procedure (Continued)

```
familymsg       db 13,10,"Processor Family: $"
period          db ".",13,10,"$"
dataCR          db ?,13,10,"$"
intel_id        db "GenuineIntel"
fpu_msg         db 13,10,"This processor contains a FPU",13,10,"$"
mce_msg         db "This processor supports the "
                db "Machine Check Exception",13,10,"$"
cmp_msg         db "This processor supports the "
                db "CMPXCHG8B instruction",13,10,"$"
not_intel       db "t least an 80486 processor.",13,10
                db "It does not contain a Genuine Intel part and as a "
                db "result,",13,10,"the CPUID detection information "
                db "cannot be determined at this time.",13,10,"$"
;
;       This code identifies the processor and coprocessor
;       that are currently in the system. The program first
;       determines the processor id. When that is accomplished,
;       the program then determines whether a coprocessor
;       exists in the system. If a coprocessor or integrated
;       coprocessor exists, the program identifies
;       the coprocessor id. The program then prints out
;       the CPU and floating point presence and type.
;
        .code
start:  mov     ax, @data
        mov     ds, ax          ; set segment register
        mov     es, ax          ; set segment register
        pushf                   ; save for restoration at end
        call    get_cpuid
        call    get_fpuid
        call    print
        popf
        mov     ax, 4c00h       ; terminate program
        int     21h

get_cpuid proc
;
;       This procedure determines the type of CPU in a system
;       and sets the cpu_type variable with the appropriate value.
;       All registers are used by this procedure, none are preserved.

;       Intel 8086 CPU check
;       Bits 12-15 of the FLAGS register are always set on the
;       8086 processor.
;
```

Example 1. CPU Identification Procedure (Continued)

```
check_8086:
        pushf                   ; push original FLAGS
        pop     ax              ; get original FLAGS
        mov     cx, ax          ; save original FLAGS
        and     ax, 0fffh       ; clear bits 12-15 in FLAGS
        push    ax              ; save new FLAGS value on stack
        popf                    ; replace current FLAGS value
        pushf                   ; get new FLAGS
        pop     ax              ; store new FLAGS in AX
        and     ax, 0f000h      ; if bits 12-15 are set, then CPU
        cmp     ax, 0f000h      ;   is an 8086/8088
        mov     cpu_type, 0     ; turn on 8086/8088 flag
        je      end_get_cpuid   ; jump if CPU is 8086/8088

;       Intel 286 CPU check
;       Bits 12-15 of the FLAGS register are always clear on the
;       Intel 286 processor in real-address mode.
;
check_80286:
        or      cx, 0f000h      ; try to set bits 12-15
        push    cx              ; save new FLAGS value on stack
        popf                    ; replace current FLAGS value
        pushf                   ; get new FLAGS
        pop     ax              ; store new FLAGS in AX
        and     ax, 0f000h      ; if bits 12-15 clear, CPU=80286
        mov     cpu_type, 2     ; turn on 80286 flag
        jz      end_get_cpuid   ; if no bits set, CPU is 80286

;       Intel386 CPU check
;       The AC bit, bit #18, is a new bit introduced in the EFLAGS
;       register on the Intel486 DX CPU to generate alignment faults.
;       This bit cannot be set on the Intel386 CPU.
;
check_80386:
;       It is now safe to use 32-bit opcode/operands
        mov     bx, sp          ; save current stack pointer to align
        and     sp, not 3       ; align stack to avoid AC fault
        OPND32
        pushf                   ; push original EFLAGS
        OPND32
        pop     ax              ; get original EFLAGS
        OPND32
        mov     cx, ax          ; save original EFLAGS
        OPND32  35h, 40000h     ; flip (XOR) AC bit in EFLAGS
        OPND32
        push    ax              ; save new EFLAGS value on stack
        OPND32
        popf                    ; replace current EFLAGS value
        OPND32
        pushf                   ; get new EFLAGS
        OPND32
        pop     ax              ; store new EFLAGS in EAX
        OPND32
        xor     ax, cx          ; can't toggle AC bit, CPU=80386
        mov     cpu_type, 3     ; turn on 80386 CPU flag
        mov     sp, bx          ; restore original stack pointer
        jz      end_get_cpuid   ; jump if 80386 CPU
        and     sp, not 3       ; align stack to avoid AC fault
```

Example 1. CPU Identification Procedure (Continued)

```
        OPND32
        push    cx
        OPND32
        popf                    ; restore AC bit in EFLAGS first
        mov     sp, bx          ; restore original stack pointer

;       Intel486 DX CPU, Intel487 SX NDP, and Intel486 SX CPU check
;       Checking for ability to set/clear ID flag (Bit 21) in EFLAGS
;       which indicates the presence of a processor
;       with the ability to use the CPUID instruction.
;
check_80486:
        mov     cpu_type, 4     ; turn on 80486 CPU flag
        OPND32
        mov     ax, cx          ; get original EFLAGS
        OPND32  35h, 200000h    ; flip (XOR) ID bit in EFLAGS
        OPND32
        push    ax              ; save new EFLAGS value on stack
        OPND32
        popf                    ; replace current EFLAGS value
        OPND32
        pushf                   ; get new EFLAGS
        OPND32
        pop     ax              ; store new EFLAGS in EAX
        OPND32
        xor     ax, cx          ; can't toggle ID bit,
        je      end_get_cpuid   ;   CPU=80486

;       Execute CPUID instruction to determine vendor, family,
;       model and stepping.
;
check_vendor:
        mov     id_flag, 1      ; set flag indicating use of CPUID inst.
        OPND32
        xor     ax, ax          ; set up input for CPUID instruction
        CPUID                   ; macro for CPUID instruction
        OPND32
        mov     word ptr vendor_id, bx  ; setup to test for vendor id
        OPND32
        mov     word ptr vendor_id[+4], dx
        OPND32
        mov     word ptr vendor_id[+8], cx
        mov     si, offset vendor_id
        mov     di, offset intel_id
        mov     cx, length intel_id
compare:
        repe    cmpsb           ; compare vendor id to "GenuineIntel"
        or      cx, cx
        jnz     end_get_cpuid   ; if not zero, not an Intel CPU,

intel_processor:
        mov     intel_proc, 1
```

Example 1. CPU Identification Procedure (Continued)

```
cpuid_data:
        OPND32
        cmp             ax, 1    ; make sure 1 is a valid input
                                 ; value for CPUID
        jl      end_get_cpuid    ; if not, jump to end
        OPND32
        xor     ax, ax           ; otherwise, use as input to CPUID
        OPND32
        inc     ax               ; and get stepping, model and family
        CPUID
        mov     stepping, al
        and     stepping, STEPPING_MASK ; isolate stepping info
        and     al, MODEL_MASK   ; isolate model info
        shr     al, MODEL_SHIFT
        mov     model, al

        and     ax, FAMILY_MASK ; mask everything but family
        shr     ax, FAMILY_SHIFT
        mov     cpu_type, al     ; set cpu_type with family

        OPND32
        mov     feature_flags, dx save feature flag data

end_get_cpuid:
        ret
get_cpuid endp

;*****************************************************************

get_fpuid proc
;
;       This procedure determines the type of FPU in a system
;       and sets the fpu_type variable with the appropriate value.
;       All registers are used by this procedure, none are preserved.

;       Coprocessor check
;       The algorithm is to determine whether the floating-point
;       status and control words can be written to. If not, no
;       coprocessor exists. If the status and control words can be
;       written to, the correct coprocessor is then determined
;       depending on the processor id. The Intel386 CPU can
;       work with either an Intel287 NDP or an Intel387 NDP.
;       The infinity of the coprocessor must be
;       checked to determine the correct coprocessor id.

        fninit                   ; reset FP status word
        mov     fp_status, 5a5ah; initialize temp word to
                                 ; non-zero value
        fnstsw  fp_status        ; save FP status word
        mov     ax, fp_status    ; check FP status word
        cmp     al, 0            ; see if correct status with
                                 ; written
        mov     fpu_type, 0      ; no fpu present
        jne     end_get_fpuid
```

Example 1. CPU Identification Procedure (Continued)

```
check_control_word:
        fnstcw  fp_status       ; save FP control word
        mov     ax, fp_status   ; check FP control word
        and     ax, 103fh       ; see if selected parts
                                ; looks OK
        cmp     ax, 3fh         ; check that 1's & 0's
                                ; correctly read
        mov     fpu_type, 0
        jne     end_get_fpuid
        mov     fpu_type, 1

;
;   80287/80387 check for the Intel386 CPU
;
check_infinity:
        cmp     cpu_type, 3
        jne     end_get_fpuid
        fldl                    ; must use default control from FNINIT
        fldz                    ; form infinity
        fdiv                    ; 8087 and Intel287 NDP say +inf = -inf
        fld     st              ; form negative infinity
        fchs                    ; Intel387 NDP says +inf <> -inf
        fcompp                  ; see if they are the same and remove them
        fstsw   fp_status       ; look at status from FCOMPP
        mov     ax, fp_status
        mov     fpu_type, 2     ; store Intel287 NDP for fpu type
        sahf                    ; see if infinities matched
        jz      end_get_fpuid   ; jump if 8087 or Intel287 is present
        mov     fpu_type, 3     ; store Intel387 NDP for fpu type
end_get_fpuid:
        ret
get_fpuid endp

;*************************************************************************

print proc
;
;       This procedure prints the appropriate cpuid string and
;       numeric processor presence status. If the CPUID instruction
;       was supported, this procedure prints out cpuid info.
;       All registers are used by this procedure, none are preserved.

        cmp     id_flag, 1      ; if set to 1, cpu supports
                                ;   CPUID instruction
                                ; print detailed CPUID information
        je      print_cpuid_data
        mov     dx, offset id_msg print initial message
        mov     ah, 9h
        int     21h

print_86:
        cmp     cpu_type, 0
```

Example 1. CPU Identification Procedure (Continued)

```
        jne     print_286
        mov     dx, offset c8086
        mov     ah, 9h
        int     21h
        cmp     fpu_type, 0
        je      end_print
        mov     dx, offset fp_8087
        mov     ah, 9h
        int     21h
        jmp     end_print

print_286:
        cmp     cpu_type, 2
        jne     print_386
        mov     dx, offset c286
        mov     ah, 9h
        int     21h
        cmp     fpu_type, 0
        je      end_print
        mov     dx, offset fp_80287
        mov     ah, 9h
        int     21h
        jmp     end_print

print_386:
        cmp     cpu_type, 3
        jne     print_486
        mov     dx, offset c386
        mov     ah, 9h
        int     21h
        cmp     fpu_type, 0
        je      end_print
        cmp     fpu_type, 2
        jne     print_387
        mov     dx, offset fp_80287
        mov     ah, 9h
        int     21h
        jmp     end_print

print_387:
        mov     dx, offset fp_80387
        mov     ah, 9h
        int     21h
        jmp     end_print

print_486:
        cmp     fpu_type, 0
        je      print_Intel486sx
        mov     dx, offset c486
        mov     ah, 9h
        int     21h
        jmp     end_print

print_Intel486sx:
        mov     dx, offset c486nfp
        mov     ah, 9h
        int     21h
        jmp     end_print
```

Example 1. CPU Identification Procedure (Continued)

```
print_cpuid_data:

cmp_vendor:
        cmp     intel_proc, 1
        jne     not_GenuineIntel

        cmp     cpu_type, 4                     ; if cpu_type=4, print
                                                ; Intel486 CPU message
        jne     check_Pentium
        mov     dx, offset Intel486_msg
        mov     ah, 9h
        int     21h
        jmp     print_family

check_Pentium:
        cmp     cpu_type, 5                     ; if cpu_type=5, print
        jne     print_features                  ; Pentium processor message
        mov     dx, offset Pentium_msg
        mov     ah, 9h
        int     21h

print_family:
        mov     dx, offset familymsg            ; print family msg
        mov     ah, 9h
        int     21h
        mov     al, cpu_type
        mov     byte ptr dataCR, al
        add     byte ptr dataCR, 30h            ; convert to ASCII
        mov     dx, offset dataCR               ; print family info
        mov     ah, 9h
        int     21h

print_model:
        mov     dx, offset modelmsg             ; print model msg
        mov     ah, 9h
        int     21h
        mov     al, model
        mov     byte ptr dataCR, al
        add     byte ptr dataCR, 30h            ; convert to ASCII
        mov     dx, offset dataCR               ; print model info
        mov     ah, 9h
        int     21h

print_stepping:
        mov     dx, offset steppingmsg          ; print stepping msg
        mov     ah, 9h
        int     21h
        mov     al, stepping
        mov     byte ptr dataCR, al
        add     byte ptr dataCR, 30h            ; convert to ASCII
        mov     dx, offset dataCR               ; print stepping info
        mov     ah, 9h
        int     21h

print_features:
        mov     ax, feature_flags
        and     ax, FPU_FLAG                    ; check for FPU
```

Example 1. CPU Identification Procedure (Continued)

```
        jz      check_MCE
        mov     dx, offset fpu_msg
        mov     ah, 9h
        int     21h

check_MCE:
        mov     ax, feature_flags
        and     ax, MCE_FLAG                    ; check for MCE
        jz      check_CMPXCHG8B

        mov     dx, offset mce_msg
        mov     ah, 9h
        int     21h

check_CMPXCHG8B:
        mov     ax, feature_flags
        and     ax, CMPXCHG8B_FLAG              ; check for CMPXCHG8B
        jz      end_print
        mov     dx, offset cmp_msg
        mov     ah, 9h
        int     21h
        jmp     end_print

not_GenuineIntel:
        mov     dx, offset not_Intel
        mov     ah, 9h
        int     21h

end_print:

        ret
print endp

        end     start
```

Example 1. CPU Identification Procedure (Continued)