AP-485

APPLICATION NOTE

Intel Processor Identification With the CPUID Instruction

October 1994

Order Number: 241618-003

Revision	Revision History	Date
-001	Original Issue.	05/93
-002	Modified Table 2. Intel486 and Pentium Processor Signatures	10/93
-003	Updated to accomodate new processor versions. Program examples modified for ease of use, section added discussing BIOS recognition for OverDrive processors, and feature flag information updated.	09/94

Additional copies of this document or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641
or call 1-800-879-4683

CONTENTS

1.0 INTRODUCTION

As the Intel Architecture evolves, with the addition of new generations and models of processors (8086, 8088, Intel 286, Intel386™, Intel486™, and Pentium™ processors), it is essential that Intel provides an increasingly sophisticated means with which software can identify the features available on each processor. This identification mechanism has evolved in conjunction with the Intel Architecture as follows:

- Originally, Intel published code sequences that could detect minor implementation differences to identify processor generations.
- Later, with the advent of the Intel386 processor, Intel implemented processor signature identification, which provided the processor family, model, and stepping numbers to software at reset.
- As the Intel Architecture evolved, Intel extended the processor signature identification into the CPUID instruction. The CPUID instruction not only provides the processor signature, but also provides information about the features supported by and implemented on the Intel processor.

The evolution of processor identification was necessary because, as the Intel Architecture proliferates, the computing market must be able to tune processor functionality across processor generations and models that have differing sets of features. Anticipating that this trend will continue with future processor generations, the Intel Architecture implementation of the CPUID instruction is extensible.

This Application Note explains how to use the CPUID instruction in software applications, BIOS implementations, and tools. By taking advantage of the CPUID instruction, software developers can create software applications and tools that can execute compatibly across the widest range of Intel processor generations and models, past, present, and future.

1.1 Update Support

New Intel processor signature and feature bits information can be obtained from the user's manual, programmer's reference manual or appropriate documentation for a processor. In addition,Intel can provide you with updated versions of the programming examples included in this application note; contact your Intel representative for more information.

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 examples at the end of this Application Note show 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 the CPUID instruction.

The CPUID instruction can be executed multiple times, each time with a different parameter value in the EAX register. The output depends on the value in the EAX register, as specified in Table 1. To determine the highest acceptable value in the EAX register, the program should set the EAX register parameter value to 0. In this case, the CPUID instruction returns the highest value that can be recognized in the EAX register. CPUID instruction execution should always use a parameter value that is less than or equal to this highest returned value. Currently, the highest value recognized by the

CPUID instruction is 1. Future processors might recognize higher values.

The processor type, specified in bits 12 and 13, indicate whether the processor is an original OEM processor, an OverDrive processor, or is a dual processor (capable of being used in a dual processor system). Table 2 shows the processor type values that can be returned in bits 12 and 13 of the EAX register.

3.1 Vendor-ID String

If the EAX register contains a value of 0, the vendor identification string is returned in the EBX, EDX, and ECX registers. These registers contain the ASCII string `GenuineIntel`.

While any imitator of the Intel 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 processor signature are implemented as described in this document.

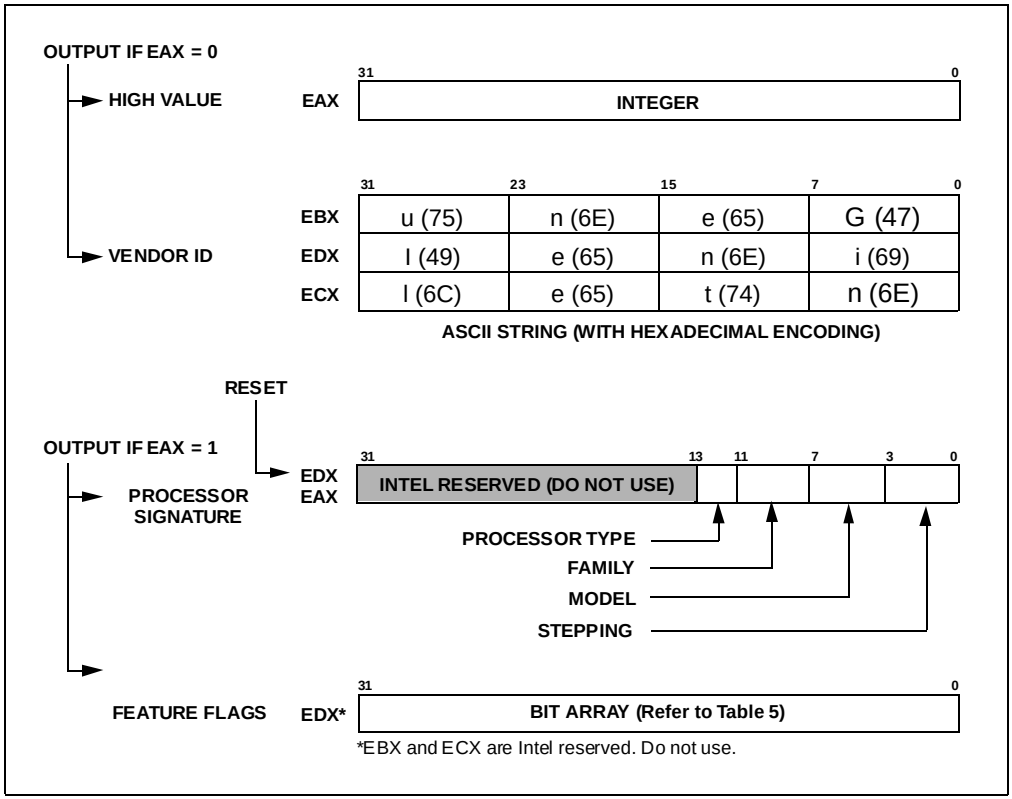

Figure 1. CPUID Instruction Outputs

Table 1. Effects of EAX Contents on CPUID Instruction Output

Parameter	Outputs of CPUID
EAX = 0	EAX ← Highest value recognized
	EBX:EDX:ECX ← Vendor identification string
EAX = 1	EAX ← Processor signature
	EDX ← Feature flags
	EBX:ECX ← Intel reserved (Do not use.)
1 < EAX ≤ highest value	Currently undefined
EAX > highest value	EAX:EBX:ECX:EDX ← Undefined (Do not use.)

Table 2. Processor Type

Bit Position	Value	Description
13,12	00	Original OEM processor
	01	OverDrive™ Processor
	10	Dual processor[1]
	11	Intel reserved. (Do not use.)

1. Not applicable to Intel386 and Intel486 processors

3.2 Processor Signature

Beginning with the Intel386 processor family, the processor signature has been available at reset. With processors that implement the CPUID instruction, the processor signature is available both upon reset and upon execution of the CPUID instruction. Figure 1 shows the format of the signature for the Intel486 and Pentium processor families. Table 3 shows the values that are currently defined. (The high-order 18 bits are undefined and reserved.)

Older versions of Intel486 SX, Intel486 DX and IntelDX2 processors do not support the CPUID instruction. Therefore, the processor signature is only available upon reset for these processors. Refer to the programming examples at the end of this Application Note to determine which processors support the CPUID instruction.

On Intel386 processors, the format of the processor signature is somewhat different, as Figure 2 shows. Table 4 gives the current values.

Table 3. Intel486™ and Pentium™ Processor Signatures

Family	Model	Stepping[1]	Description
0100	0000 and 0001	xxxx	Intel486 DX Processors
0100	0010	xxxx	Intel486 SX Processors
0100	0011	xxxx	Intel487™ Processors[2]
0100	0011	xxxx	IntelDX2™ and IntelDX2 OverDrive™ Processors
0100	0100	xxxx	Intel486 SL Processor[2]
0100	0101	xxxx	IntelSX2™ Processors
0100	0111	xxxx	Write-Back Enhanced IntelDX2 Processors
0100	1000	xxxx	IntelDX4™ and IntelDX4 OverDrive Processors
0101	0001	xxxx	Pentium™ Processors (510\60, 567\66)
0101	0010	xxxx	Pentium Processors (735\90, 815\100)
0101	0011	xxxx	Pentium OverDrive Processors
0101	0101	xxxx	Reserved for Pentium OverDrive Processor for IntelDX4 Processor
0101	0010	xxxx	Reserved for Pentium OverDrive Processor for Pentium Processor (510/60, 567/66)
0101	0100	xxxx	Reserved for Pentium OverDrive Processor for Pentium Processor (735\90, 815\100)

1. Intel releases information about stepping numbers as needed.
2. This processor does not implement the CPUID instruction.

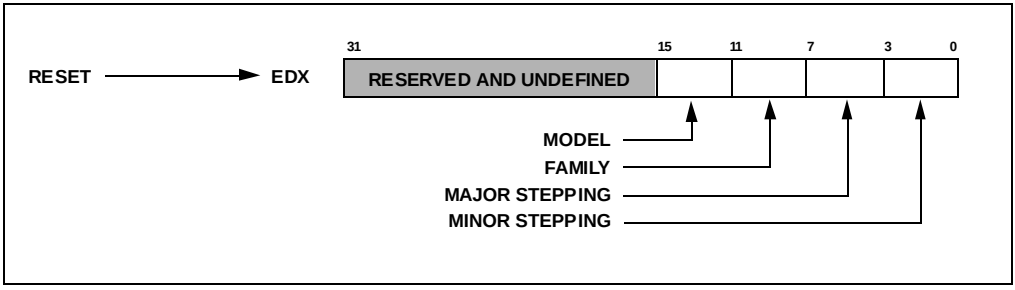

Figure 2. Processor Signature Format on Intel386™ Processors

Table 4. Intel386™ Processor Signatures

Model	Family	Major Stepping	Minor Stepping[1]	Description
0000	0011	0000	xxxx	Intel386™ DX Processor
0010	0011	0000	xxxx	Intel386 SX Processor
0010	0011	0000	xxxx	Intel386 CX Processor
0010	0011	0000	xxxx	Intel386 EX Processor
0100	0011	0000 and 0001	xxxx	Intel386 SL Processor
0000	0011	0100	xxxx	RAPIDCAD™ Processor

1. Intel releases information about minor stepping numbers as needed.

3.3 Feature Flags

When a value of 1 is placed in the EAX register, the CPUID instruction loads the EDX register with the feature flags. The feature flags indicate which features the processor supports. A value of 1 in a feature flag can indicate that a feature is either supported or not supported, depending on the implementation of the CPUID instruction for a specific processor. Table 5 lists the currently defined feature flag values. For future processors, refer to the programmer's reference manual, user's manual, or the appropriate documentation for the latest feature flag values.

Developers should use the feature flags in applications to determine which processor features are supported. By using the CPUID feature flags to predetermine processor features, software can detect and avoid incompatibilities that could result if the features are not present.

Table 5. Feature Flag Values

Bit	Name	Description When Flag = 1	Comments
0	FPU	Floating-point unit on-chip	The processor contains an FPU that supports the Intel 387 floating-point instruction set.
1	VME	Virtual Mode Extension	The processor supports extensions to virtual-8086 mode.
2[1]			(see note)
3	PSE	Page Size Extension	The processor supports 4-Mbyte pages.
4–6[1]			(see note)
7	MCE	Machine Check	Exception 18 is defined for Pentum processor style machine checks, including CR4.MCE for controlling the feature. This feature does not define the model-specific implementation of the machine -check error logging reporting and processor shutdowns. Machine-check exception handlers may have to depend on processor version to do model-specific processing of the exception or test for the presence of the standard machine-check feature.
8	CX8	CMPXCHG8B	The 8-byte (64-bit) compare and exchange instructions is supported (implicitly locked and atomic).
9	APIC	On-chip APIC	Indicates that an integrated APIC is present and hardware enabled. (Software disabling does not affect this bit.)
10–31[1]			(see note)

1. Some non-essential information regarding Intel486 and Pentium processors is considered Intel confidential and proprietary and is not 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 Intel-recommended feature-detection methods. Software should not try to identify features by exploiting programming tricks, undocumented features, or otherwise deviating from the guidelines presented in this Application Note. The following is a list of guidelines 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 6.0 .
- Do not assume that a given family or model has any specific feature. For example, do not assume that, because the family value is 5 (Pentium processor), there must be a floating-point unit on-chip. Use the feature flags for this determination.
- Do not assume that the features in the OverDrive processors are the same as those in the OEM version of the processor. Internal caches and instruction execution might vary.

- Do not use undocumented features of a processor to identify steppings or features. For example, the Intel386 processor 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 processor used the same opcodes for different instructions. That software should have used the stepping information in the processor 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 models of the Pentium processor in which the CPUID instruction is implemented. 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 test feature flags individually and not make assumptions about undefined bits. It would be a mistake, for example, to test the FPU bit by comparing the feature register to a binary 1 with a compare instruction.
- Do not assume that the clock of a given family or model runs at a specific frequency and do not write clock-dependent code, such as timing loops. For instance, an OverDrive Processor could operate at a higher internal frequency and still report the same family and/or model. Instead, use the system's timers to measure elapsed time.
- Processor model-specific registers may differ among processors, including in various models of the Pentium processor. Do not use these registers unless identified for the installed processor.

5.0 BIOS RECOGNITION FOR INTEL OVERDRIVE™ PROCESSORS

A system's BIOS will typically identify the processor in the system and initialize the hardware accordingly. In many cases, the BIOS identifies the processor by reading the processor signature, comparing it to known signatures, and, upon finding a match, executing the corresponding hardware initialization code.

The Pentium OverDrive processor is designed to be an upgrade to any Intel486 family processor. Because there are significant operational differences between these two processor families, processor misidentification can cause system failures or diminished performance. Major differences between the Intel486 processor and the Pentium OverDrive processor include the type of on-chip cache supported (write-back or write-through), cache organization and cache size. The OverDrive processor also has an enhanced floating point unit and System Management Mode (SMM) that may not exist in the OEM processor. Inability to recognize these features causes problems like those described below.

In many BIOS implementations, the BIOS reads the processor signature at reset and compares it to known values. If the OverDrive processor's signature is not among the known values, a match will not occur and the OverDrive processor will not be identified. Often the BIOS will drop out of the search and initialize the hardware based on a default case such as intializing the chipset for an Intel486 SX processor. Below are two common examples of system failures and how to avoid them.

Example 1

If (for the Pentium OverDrive processor) the system's hardware is configured to enable the write-back cache but the BIOS fails to detect the Pentium OverDrive processor signature, the BIOS may incorrectly cause the chipset to sup-

port a write-through processor cache. This results in a data incoherency problem with the bus masters. When a bus master accesses a memory location (which was also in the processor's cache in a modified state), the processor will alert the chipset to allow it to update this data in memory. But the chipset is not programmed for such an event and the bus master instead receives stale data. This usually results in a system failure.

Example 2

If the BIOS does not recognize the OverDrive processor's signature and defaults to a Intel486 SX processor, the BIOS can incorrectly program the chipset to ignore, or improperly route, the assertion of the floating point error signaled by the processor. The result is that floating point errors will be improperly handled by the Pentium OverDrive processor. The BIOS may also completely disable math exception handling in the OverDrive processor. This can cause installation errors in applications that require hardware support for floating point instructions.

Hence, when programming or modifying a BIOS, be aware of the impact of future OverDrive processors. Intel recommends that you include processor signatures for the OverDrive processors in BIOS identification routines to eliminate diminished performance or system failures. The recommendations in this application note can help a BIOS maintain compatibility across a wide range of processor generations and models.

6.0 PROPER IDENTIFICATION SEQUENCE

The `cpuid3a.asm` program example demonstrates the correct use of the CPUID instruction. (See Example 1.) It also shows how to identify earlier processor generations that do not implement the processor signature or CPUID instruction. This program example contains the following two procedures:

- **`get_cpu_type`** identifies the processor type. Figure 3 illustrates the flow of this procedure.
- **`get_fpu_type`** determines the type of floating-point unit (FPU) or math coprocessor (MCP).

This procedure has been tested with 8086, 80286, Intel386, Intel486, and Pentium processors. This program example is written in assembly language and is suitable for inclusion in a run-time library, or as system calls in operating systems.

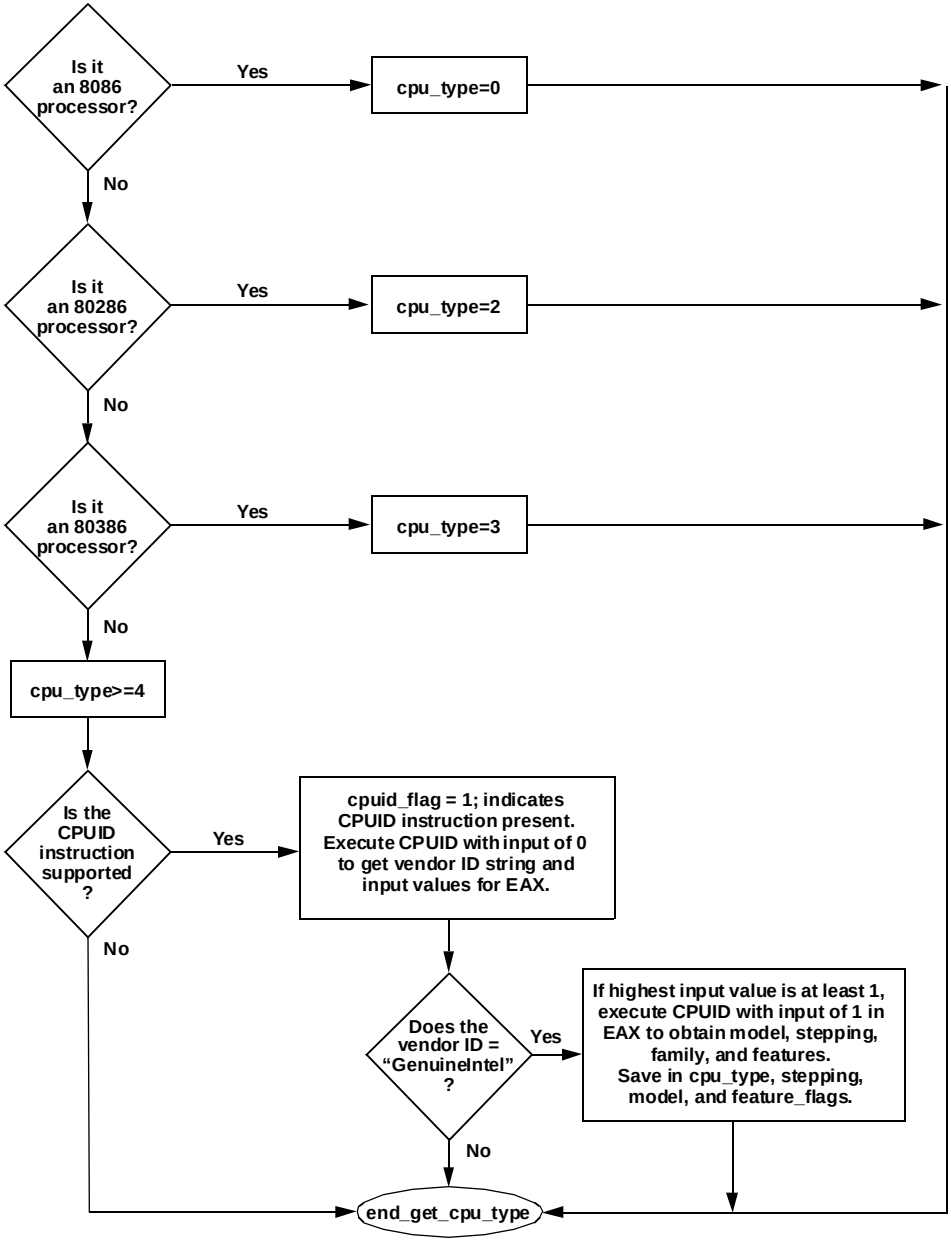

Figure 3. Flow of Processor get_cpu_type Procedure

7.0 USAGE PROGRAM EXAMPLE

The `cpuid3b.asm` and `cpuid3b.c` program examples demonstrate applications that call `get_cpu_type` and `get_fpu_type` procedures and interpret the returned information. The results, which are displayed on the monitor, identify the installed processor and features. The `cpuid3b.asm` example is written in assembly language and demonstrates an application that displays the returned information in the DOS environment. The `cpuid3b.c` example is written in the C language. (See Examples 2 and 3.)

Figure 4 presents an overview of the relationship between the three program examples.

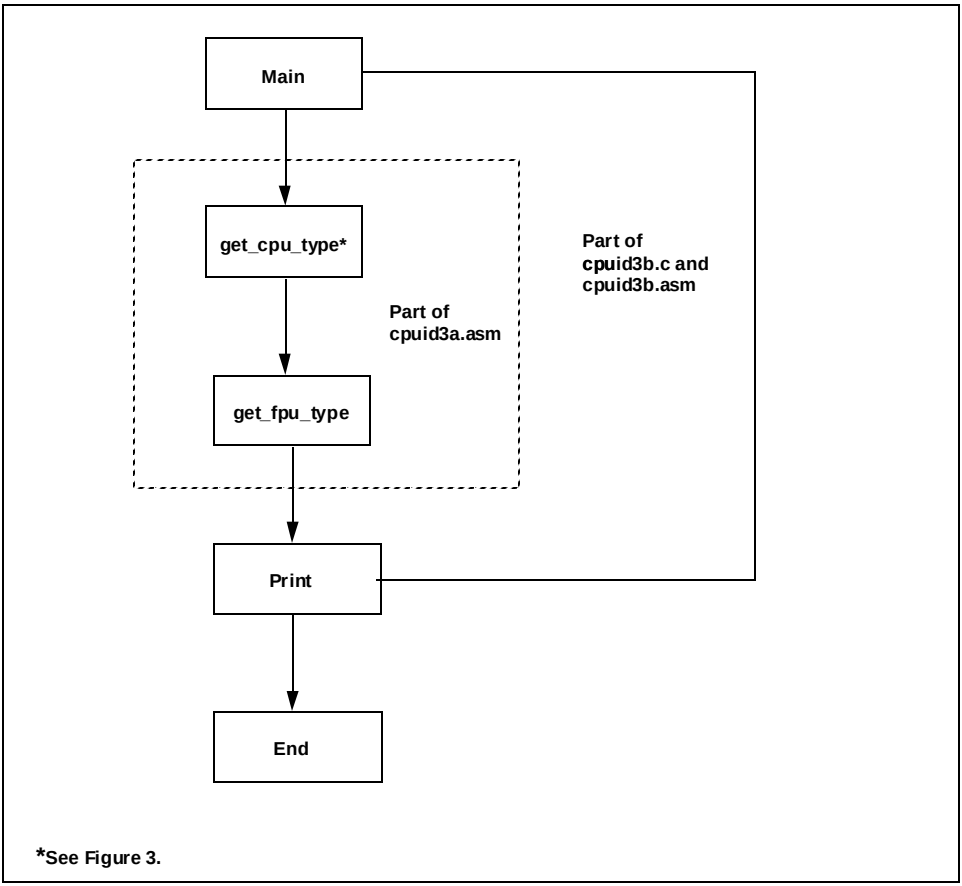

Figure 4. Flow of Processor Identification Extraction Procedures

Example 1. Processor Identification Extraction Procedure

```
;       Filename:       cpuid3a.asm
;       Copyright 1993, 1994 by Intel Corp.
;
;       This program has been developed by Intel Corporation.  You
;       have Intel's permission to incorporate this source code into
;       your product, royalty free.  Intel has intellectual property
;       rights which it may assert if another manufacturer's processor
;       mis-identifies itself as being "GenuineIntel" when the CPUID
;       instruction is executed.
;
;       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, and
;       including the warranties of merchantability and fitness for a
;       particular purpose.  Intel does not assume any responsibility
;       for any errors which may appear in this code nor any
;       responsibility to update it.
;
;       This code contains two procedures:
;       _get_cpu_type: Identifies processor type in _cpu_type:
;               0=8086/8088 processor
;               2=Intel 286 processor
;               3=Intel386(TM) family processor
;               4=Intel486(TM) family processor
;               5=Pentium(TM) family processor
;
;       _get_fpu_type: Identifies FPU type in _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)
;
;       This program has been tested with the MASM assembler.
;       This code correctly detects the current Intel 8086/8088,
;       80286, 80386, 80486, and Pentium(tm) processors in the
;       real-address mode.
;
;       To assemble this code with TASM, add the JUMPS directive.
;       jumps                   ; Uncomment this line for TASM


        TITLE   cpuid3a
        DOSSEG
        .model  small

CPU_ID  MACRO
```

```
        db      0fh             ; Hardcoded CPUID instruction
        db      0a2h
ENDM

        .data
        public  _cpu_type
        public  _fpu_type
        public  _cpuid_flag
        public  _intel_CPU
        public  _vendor_id
        public  _cpu_signature
        public  _features_ecx
        public  _features_edx
        public  _features_ebx
_cpu_type       db      0
_fpu_type       db      0
_cpuid_flag     db      0
_intel_CPU      db      0
_vendor_id      db      "------------"
intel_id        db      "GenuineIntel"
_cpu_signature  dd      0
_features_ecx   dd      0
_features_edx   dd      0
_features_ebx   dd      0
fp_status       dw      0

        .code
        .8086

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

        public  _get_cpu_type
_get_cpu_type   proc

;       This procedure determines the type of processor in a system
;       and sets the _cpu_type variable with the appropriate
;       value.  If the CPUID instruction is available, it is used
;       to determine more specific details about the processor.
;       All registers are used by this procedure, none are preserved.
;       To avoid AC faults, the AM bit in CR0 must not be set.

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

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
        cmp     ax, 0f000h      ;   processor is an 8086/8088
        mov     _cpu_type, 0    ; turn on 8086/8088 flag
        je      end_cpu_type    ; jump if processor is 8086/8088

;       Intel 286 processor check
;       Bits 12-15 of the FLAGS register are always clear on the
;       Intel 286 processor in real-address mode.

        .286
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 are clear
        mov     _cpu_type, 2    ; processor=80286, turn on 80286 flag
        jz      end_cpu_type    ; if no bits set, processor is 80286

;       Intel386 processor check
;       The AC bit, bit #18, is a new bit introduced in the EFLAGS
;       register on the Intel486 processor to generate alignment
;       faults.
;       This bit cannot be set on the Intel386 processor.

        .386                    ; it is safe to use 386 instructions
check_80386:
        pushfd                  ; push original EFLAGS
        pop     eax             ; get original EFLAGS
        mov     ecx, eax        ; save original EFLAGS
        xor     eax, 40000h     ; flip AC bit in EFLAGS
        push    eax             ; save new EFLAGS value on stack
        popfd                   ; replace current EFLAGS value
        pushfd                  ; get new EFLAGS
        pop     eax             ; store new EFLAGS in EAX
        xor     eax, ecx        ; can't toggle AC bit, processor=80386
        mov     _cpu_type, 3    ; turn on 80386 processor flag
        jz      end_cpu_type    ; jump if 80386 processor

        push    ecx
        popfd                   ; restore AC bit in EFLAGS first

;       Intel486 processor check
;       Checking for ability to set/clear ID flag (Bit 21) in EFLAGS
;       which indicates the presence of a processor with the CPUID
```

```
;       instruction.

        .486
check_80486:
        mov     _cpu_type, 4    ; turn on 80486 processor flag
        mov     eax, ecx        ; get original EFLAGS
        xor     eax, 200000h    ; flip ID bit in EFLAGS
        push    eax             ; save new EFLAGS value on stack
        popfd                   ; replace current EFLAGS value
        pushfd                  ; get new EFLAGS
        pop     eax             ; store new EFLAGS in EAX
        xor     eax, ecx        ; can't toggle ID bit,
        je      end_cpu_type    ; processor=80486

;       Execute CPUID instruction to determine vendor, family,
;       model, stepping and features.  For the purpose of this
;       code, only the initial set of CPUID information is saved.

        mov     _cpuid_flag, 1  ; flag indicating use of CPUID inst.
        push    ebx             ; save registers
        push    esi
        push    edi
        mov     eax, 0          ; set up for CPUID instruction
        CPU_ID                  ; get and save vendor ID

        mov     dword ptr _vendor_id, ebx
        mov     dword ptr _vendor_id[+4], edx
        mov     dword ptr _vendor_id[+8], ecx

        mov     si, ds
        mov     es, si

        mov     si, offset _vendor_id
        mov     di, offset intel_id
        mov     cx, 12          ; should be length intel_id
        cld                     ; set direction flag
        repe    cmpsb           ; compare vendor ID to "GenuineIntel"
        jne     end_cpuid_type  ; if not equal, not an Intel processor

        mov     _intel_CPU, 1   ; indicate an Intel processor
        cmp     eax, 1          ; make sure 1 is valid input for CPUID
        jl      end_cpuid_type  ; if not, jump to end
        mov     eax, 1
        CPU_ID                  ; get family/model/stepping/features
        mov     _cpu_signature, eax
        mov     _features_ebx, ebx
        mov     _features_edx, edx
        mov     _features_ecx, ecx

        shr     eax, 8          ; isolate family
```

```
        and     eax, 0fh
        mov     _cpu_type, al   ; set _cpu_type with family

end_cpuid_type:
        pop     edi             ; restore registers
        pop     esi
        pop     ebx

        .8086
end_cpu_type:
        ret
_get_cpu_type   endp

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

        public  _get_fpu_type
_get_fpu_type   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 are present.  If not, no
;       coprocessor exists.  If the status and control words can
;       be saved, the correct coprocessor is then determined
;       depending on the processor type.  The Intel386 processor can
;       work with either an Intel287 NDP or an Intel387 NDP.
;       The infinity of the coprocessor must be checked to determine
;       the correct coprocessor type.

        fninit                  ; reset FP status word
        mov     fp_status, 5a5ah; initialize temp word to non-zero
        fnstsw  fp_status       ; save FP status word
        mov     ax, fp_status   ; check FP status word
        cmp     al, 0           ; was correct status written
        mov     _fpu_type, 0    ; no FPU present
        jne     end_fpu_type

check_control_word:
        fnstcw  fp_status       ; save FP control word
        mov     ax, fp_status   ; check FP control word
        and     ax, 103fh       ; selected parts to examine
        cmp     ax, 3fh         ; was control word correct
        mov     _fpu_type, 0
        jne     end_fpu_type    ; incorrect control word, no FPU
        mov     _fpu_type, 1

;       80287/80387 check for the Intel386 processor
```

```
check_infinity:
        cmp     _cpu_type, 3
        jne     end_fpu_type
        fld1                    ; must use default control from FNINIT
        fldz                    ; form infinity
        fdiv                    ; 8087/Intel287 NDP say +inf = -inf
        fld     st              ; form negative infinity
        fchs                    ; Intel387 NDP says +inf <> -inf
        fcompp                  ; see if they are the same
        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_fpu_type    ; jump if 8087 or Intel287 is present
        mov     _fpu_type, 3    ; store Intel387 NDP for FPU type
end_fpu_type:
        ret
_get_fpu_type   endp

        end
```

Example 2. Processor Identification Procedure in Assembly Language

```
;       Filename:       cpuid3b.asm
;       Copyright 1993, 1994 by Intel Corp.
;
;       This program has been developed by Intel Corporation.  You
;       have Intel's permission to incorporate this source code into
;       your product, royalty free.  Intel has intellectual property
;       rights which it may assert if another manufacturer's processor
;       mis-identifies itself as being "GenuineIntel" when the CPUID
;       instruction is executed.
;
;       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, and
;       including the warranties of merchantability and fitness for a
;       particular purpose.  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 processor type in the variable _cpu_type:
;
;       Part 2: Identifies FPU type in the variable _fpu_type:
;
;       Part 3: Prints out the appropriate message.  This part is
;               specific to the DOS environment and uses the DOS
;               system calls to print out the messages.
;
;       This program has been tested with the MASM assembler.
;       If this code is assembled with no options specified and linked
;       with the cpuid3a module, it correctly identifies the
;       current Intel 8086/8088, 80286, 80386, 80486, and Pentium(tm)
;       processors in the real-address mode.
;
;       To assemble this code with TASM, add the JUMPS directive.
;       jumps                   ; Uncomment this line for TASM


        TITLE   cpuid3b
        DOSSEG
        .model  small
        .stack  100h

        .data
        extrn   _cpu_type: byte
        extrn   _fpu_type: byte
        extrn   _cpuid_flag: byte
```

```
        extrn   _intel_CPU: byte
        extrn   _vendor_id: byte
        extrn   _cpu_signature: dword
        extrn   _features_ecx: dword
        extrn   _features_edx: dword
        extrn   _features_ebx: dword

;       The purpose of this code is to identify the processor and
;       coprocessor that is currently in the system.  The program
;       first determines the processor type.  Then it determines
;       whether a coprocessor exists in the system.  If a
;       coprocessor or integrated coprocessor exists, the program
;       identifies the coprocessor type.  The program then prints
;       the processor and floating point processors present and type.

        .code
        .8086
start:  mov     ax, @data
        mov     ds, ax          ; set segment register
        mov     es, ax          ; set segment register
        and     sp, not 3       ; align stack to avoid AC fault
        call    _get_cpu_type   ; determine processor type
        call    _get_fpu_type
        call    print
        mov     ax, 4c00h       ; terminate program
        int     21h

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

        extrn   _get_cpu_type: proc

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

        extrn   _get_fpu_type: proc

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


FPU_FLAG        equ     0001h
VME_FLAG        equ     0002h
PSE_FLAG        equ     0008h
MCE_FLAG        equ     0080h
CMPXCHG8B_FLAG  equ     0100h
APIC_FLAG       equ     0200h

        .data
id_msg          db      "This system has a$"
cp_error        db      "n unknown processor$"
cp_8086         db      "n 8086/8088 processor$"
cp_286          db      "n 80286 processor$"
```

```
cp_386          db      "n 80386 processor$"

cp_486          db      "n 80486DX, 80486DX2 processor or"
                db      " 80487SX math coprocessor$"
cp_486sx        db      "n 80486SX processor$"

fp_8087         db      " and an 8087 math coprocessor$"
fp_287          db      " and an 80287 math coprocessor$"
fp_387          db      " and an 80387 math coprocessor$"

intel486_msg    db      " Genuine Intel486(TM) processor$"
intel486dx_msg  db      " Genuine Intel486(TM) DX processor$"
intel486sx_msg  db      " Genuine Intel486(TM) SX processor$"
inteldx2_msg    db      " Genuine IntelDX2(TM) processor$"
intelsx2_msg    db      " Genuine IntelSX2(TM) processor$"
inteldx4_msg    db      " Genuine IntelDX4(TM) processor$"
inteldx2wb_msg  db      " Genuine Write-Back Enhanced"
                db      " IntelDX2(TM) processor$"
pentium_msg     db      " Genuine Intel Pentium(TM) processor$"
unknown_msg     db      "n unknown Genuine Intel processor$"

; The following 16 entries must stay intact as an array
intel_486_0     dw      offset intel486dx_msg
intel_486_1     dw      offset intel486dx_msg
intel_486_2     dw      offset intel486sx_msg
intel_486_3     dw      offset inteldx2_msg
intel_486_4     dw      offset intel486_msg
intel_486_5     dw      offset intelsx2_msg
intel_486_6     dw      offset intel486_msg
intel_486_7     dw      offset inteldx2wb_msg
intel_486_8     dw      offset inteldx4_msg
intel_486_9     dw      offset intel486_msg
intel_486_a     dw      offset intel486_msg
intel_486_b     dw      offset intel486_msg
intel_486_c     dw      offset intel486_msg
intel_486_d     dw      offset intel486_msg
intel_486_e     dw      offset intel486_msg
intel_486_f     dw      offset intel486_msg
; end of array

family_msg      db      13,10,"Processor Family:  $"
model_msg       db      13,10,"Model:             $"
stepping_msg    db      13,10,"Stepping:          "
cr_lf           db      13,10,"$"

turbo_msg       db      13,10,"The processor is an OverDrive(TM)"
                db      " processor$"
dp_msg          db      13,10,"The processor is the upgrade processor"
                db      " in a dual processor system$"
fpu_msg         db      13,10,"The processor contains an on-chip FPU$"
```

```
mce_msg         db      13,10,"The processor supports Machine Check"
                db      " Exceptions$"
cmp_msg         db      13,10,"The processor supports the CMPXCHG8B"
                db      " instruction$"
vme_msg         db      13,10,"The processor supports Virtual Mode"
                db      " Extensions$"
pse_msg         db      13,10,"The processor supports Page Size"
                db      " Extensions$"
apic_msg        db      13,10,"The processor contains an on-chip"
                db      " APIC$"

not_intel       db      "t least an 80486 processor."
                db      13,10,"It does not contain a Genuine Intel"
                db      " part and as a result, the",13,10,"CPUID"
                db      " detection information cannot be determined"
                db      " at this time.$"

ASC_MSG MACRO   msg
        LOCAL   ascii_done              ; local label
        add     al, 30h
        cmp     al, 39h                 ; is it 0-9?
        jle     ascii_done
        add     al, 07h
ascii_done:
        mov     byte ptr msg[20], al
        mov     dx, offset msg
        mov     ah, 9h
        int     21h
ENDM

        .code
        .8086
print   proc

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

        mov     dx, offset id_msg       ; print initial message
        mov     ah, 9h
        int     21h

        cmp     _cpuid_flag, 1          ; if set to 1, processor
                                        ;   supports CPUID instruction
        je      print_cpuid_data        ; print detailed CPUID info

print_86:
        cmp     _cpu_type, 0
        jne     print_286
```

```
        mov     dx, offset cp_8086
        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 cp_286
        mov     ah, 9h
        int     21h
        cmp     _fpu_type, 0
        je      end_print
print_287:
        mov     dx, offset fp_287
        mov     ah, 9h
        int     21h
        jmp     end_print

print_386:
        cmp     _cpu_type, 3
        jne     print_486
        mov     dx, offset cp_386
        mov     ah, 9h
        int     21h
        cmp     _fpu_type, 0
        je      end_print
        cmp     _fpu_type, 2
        je      print_287
        mov     dx, offset fp_387
        mov     ah, 9h
        int     21h
        jmp     end_print

print_486:
        cmp     _cpu_type, 4
        jne     print_unknown           ; Intel processors will have
        mov     dx, offset cp_486sx     ; CPUID instruction
        cmp     _fpu_type, 0
        je      print_486sx
        mov     dx, offset cp_486
print_486sx:
        mov     ah, 9h
        int     21h
        jmp     end_print
```

```
print_unknown:
        mov     dx, offset cp_error
        jmp     print_486sx

print_cpuid_data:
        .486
        cmp     _intel_CPU, 1           ; check for genuine Intel
        jne     not_GenuineIntel        ;   processor
print_486_type:
        cmp     _cpu_type, 4            ; if 4, print 80486 processor
        jne     print_pentium_type
        mov     ax, word ptr _cpu_signature
        shr     ax, 4
        and     eax, 0fh                ; isolate model
        mov     dx, intel_486_0[eax*2]
        jmp     print_common
print_pentium_type:
        cmp     _cpu_type, 5           ; if 5, print Pentium processor
        jne     print_unknown_type
        mov     dx, offset pentium_msg
        jmp     print_common
print_unknown_type:
        mov     dx, offset unknown_msg  ; if neither, print unknown

print_common:
        mov     ah, 9h
        int     21h

; print family, model, and stepping

print_family:
        mov     al, _cpu_type
        ASC_MSG family_msg              ; print family msg

print_model:
        mov     ax, word ptr _cpu_signature
        shr     ax, 4
        and     al, 0fh
        ASC_MSG model_msg               ; print model msg

print_stepping:
        mov     ax, word ptr _cpu_signature
        and     al, 0fh
        ASC_MSG stepping_msg            ; print stepping msg

print_upgrade:
        mov     ax, word ptr _cpu_signature
        test    ax, 1000h               ; check for turbo upgrade
        jz      check_dp
```

```
        mov     dx, offset turbo_msg
        mov     ah, 9h
        int     21h
        jmp     print_features

check_dp:
        test    ax, 2000h               ; check for dual processor
        jz      print_features
        mov     dx, offset dp_msg
        mov     ah, 9h
        int     21h

print_features:
        mov     ax, word ptr _features_edx
        and     ax, FPU_FLAG            ; check for FPU
        jz      check_MCE
        mov     dx, offset fpu_msg
        mov     ah, 9h
        int     21h

check_MCE:
        mov     ax, word ptr _features_edx
        and     ax, MCE_FLAG            ; check for MCE
        jz      check_CMPXCHG8B
        mov     dx, offset mce_msg
        mov     ah, 9h
        int     21h

check_CMPXCHG8B:
        mov     ax, word ptr _features_edx
        and     ax, CMPXCHG8B_FLAG      ; check for CMPXCHG8B
        jz      check_VME
        mov     dx, offset cmp_msg
        mov     ah, 9h
        int     21h

check_VME:
        mov     ax, word ptr _features_edx
        and     ax, VME_FLAG            ; check for VME
        jz      check_PSE
        mov     dx, offset vme_msg
        mov     ah, 9h
        int     21h

check_PSE:
        mov     ax, word ptr _features_edx
        and     ax, PSE_FLAG            ; check for PSE
        jz      check_APIC
        mov     dx, offset pse_msg
        mov     ah, 9h
```

```
        int     21h

check_APIC:
        mov     ax, word ptr _features_edx
        and     ax, APIC_FLAG           ; check for APIC
        jz      end_print
        mov     dx, offset apic_msg
        mov     ah, 9h
        int     21h

        jmp     end_print

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

end_print:
        mov     dx, offset cr_lf
        mov     ah, 9h
        int     21h
        ret
print   endp

        end     start
```

Example 3. Processor Identification Procedure in the C Language

```
/* Filename:    cpuid3b.c                                          */
/* Copyright 1994 by Intel Corp.                                   */
/*                                                                 */
/* This program has been developed by Intel Corporation.  You     */
/* have Intel's permission to incorporate this source code into   */
/* your product, royalty free. Intel has intellectual property    */
/* rights which it may assert if another manufacturer's processor*/
/* mis-identifies itself as being "GenuineIntel" when the CPUID   */
/* instruction is executed.                                       */
/*                                                                 */
/* 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, and      */
/* including the warranties of merchantability and fitness for a */
/* particular purpose.  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:         */
/*                                                                 */
/* Part 2: Identifies FPU type in the variable _fpu_type:         */
/*                                                                 */
/* Part 3: Prints out the appropriate message.                    */
/*                                                                 */
/* This program has been tested with the Microsoft C compiler.    */
/* If this code is compiled with no options specified and linked */
/* with the cpuid3a module, it correctly identifies the           */
/* current Intel 8086/8088, 80286, 80386, 80486, and              */
/* Pentium(tm) processors in the real-address mode.               */

#define FPU_FLAG        0x0001
#define VME_FLAG        0x0002
#define PSE_FLAG        0x0008
#define MCE_FLAG        0x0080
#define CMPXCHG8B_FLAG  0x0100
#define APIC_FLAG       0x0200

extern char cpu_type;
extern char fpu_type;
extern char cpuid_flag;
extern char intel_CPU;
extern char vendor_id[12];
extern long cpu_signature;
extern long features_ecx;
extern long features_edx;
```

```
extern long features_ebx;
main() {
    get_cpu_type();
    get_fpu_type();
    print();
}
print() {
    printf("This system has a");
    if (cpuid_flag == 0) {
        switch (cpu_type) {
        case 0:
            printf("n 8086/8088 processor");
            if (fpu_type) printf(" and an 8087 math coprocessor");
            break;
        case 2:
            printf("n 80286 processor");
            if (fpu_type) printf(" and an 80287 math coprocessor");
            break;
        case 3:
            printf("n 80386 processor");
            if (fpu_type == 2)
                printf(" and an 80287 math coprocessor");
            else if (fpu_type)
                printf(" and an 80387 math coprocessor");
            break;
        case 4:
            if (fpu_type) printf("n 80486DX, 80486DX2 processor or \
80487SX math coprocessor");
            else printf("n 80486SX processor");
            break;
        default:
            printf("n unknown processor");
        }
    } else {
        /* using cpuid instruction */
        if (intel_CPU) {
            if (cpu_type == 4) {
                switch ((cpu_signature>>4)&0xf) {
                case  0:
                case  1:
                    printf(" Genuine Intel486(TM) DX processor");
                    break;
                case  2:
                    printf(" Genuine Intel486(TM) SX processor");
                    break;
                case  3:
                    printf(" Genuine IntelDX2(TM) processor");
                    break;
                case  4:
                    printf(" Genuine Intel486(TM) processor");
```

```
                    break;
                case  5:
                    printf(" Genuine IntelSX2(TM) processor");
                    break;
                case  7:
                    printf(" Genuine Write-Back Enhanced \
IntelDX2(TM) processor");
                    break;
                case  8:
                    printf(" Genuine IntelDX4(TM) processor");
                    break;
                default:
                    printf(" Genuine Intel486(TM) processor");
                }
            } else if (cpu_type == 5)
                printf(" Genuine Intel Pentium(TM) processor");
            else
                printf("n unknown Genuine Intel processor");
            printf("\nProcessor Family: %X", cpu_type);
            printf("\nModel:            %X", (cpu_signature>>4)&0xf);
            printf("\nStepping:         %X\n", cpu_signature&0xf);
            if (cpu_signature & 0x1000)
                printf("\nThe processor is an OverDrive(TM) upgrade
          \processor");
            else if (cpu_signature & 0x2000)
                printf("\nThe processor is the upgrade processor \
in a dual processor system");
            if (features_edx & FPU_FLAG)
                printf("\nThe processor contains an on-chip FPU");
            if (features_edx & MCE_FLAG)
                printf("\nThe processor supports Machine Check \
Exceptions");
            if (features_edx & CMPXCHG8B_FLAG)
                printf("\nThe processor supports the CMPXCHG8B \
instruction");
            if (features_edx & VME_FLAG)
                printf("\nThe processor supports Virtual Mode \
Extensions");
            if (features_edx & PSE_FLAG)
                printf("\nThe processor supports Page Size \
Extensions");
            if (features_edx & APIC_FLAG)
                printf("\nThe processor contains an on-chip APIC");
        } else {
            printf("t least an 80486 processor.\nIt does not \
contain a Genuine Intel part and as a result, the\nCPUID detection \
information cannot be determined at this time.");
        }
    }
    printf("\n");
}
```