



AP-485

**APPLICATION
NOTE**

**Intel Processor Identification
With the CPUID Instruction**

December 1995

Order Number: 241618-004



Revision	Revision History	Date
-001	Original Issue.	05/93
-002	Modified Table 2. Intel486 and Pentium® Processor Signatures	10/93
-003	Updated to accommodate 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
-004	Updated with Pentium Pro and OverDrive processors information. Modified Tables 1, 3 and 5. Inserted Tables 6, 7 and 8. Inserted Section 3.4. and 3.5.	12/95

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

Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer products may have minor variations to this specification, known as errata.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

MDS is an ordering code only and is not used as a product name or trademark of Intel Corporation.

Intel Corporation and Intel's FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

Since publication of documents referenced in this document, registration of the Pentium and iCOMP trademarks has been issued to Intel Corporation.

* Other brands and names are the property of their respective owners.

Copyright © 1993 -1995, Intel Corporation. All rights reserved.

CG/042193.

CONTENTS

	PAGE		PAGE
1.0 INTRODUCTION	1	Examples	
1.1 Update Support	1	Example 1. Processor Identification Extraction Procedure	12
2.0 DETECTING THE CPUID INSTRUCTION	1	Example 2. Processor Identification Procedure in Assembly Language	18
3.0 OUTPUTS OF THE CPUID INSTRUCTION	1	Example 3. Processor Identification Procedure in the C Language	28
3.1 Vendor-ID String	2	Figures	
3.2 Processor Signature	3	Figure 1. CPUID Instruction Outputs	2
3.3 Feature Flags	5	Figure 2. Processor Signature Format on Intel386 Processors	5
3.4 Cache Size and Format Information	7	Figure 3. Flow of Processor get_cpu_type Procedure	10
3.5 Example	8	Figure 4. Flow of Processor Identification Extraction Procedures	11
4.0 USAGE GUIDELINES	8	Tables	
5.0 BIOS RECOGNITION FOR INTEL OVERDRIVE[®] PROCESSORS	9	Table 1. Effects of EAX Contents on CPUID Instruction Output	3
6.0 PROPER IDENTIFICATION SEQUENCE	11	Table 2. Processor Type	3
		Table 3. Intel486 [™] Pentium [®] Processor, and Pentium Pro Processor Signatures	4
		Table 4. Intel386 [™] Processor Signatures	5
		Table 5. Feature Flag Values	6
		Table 6. Descriptor Formats	7
		Table 7. Descriptor Decode Values	7
		Table 8. Pentium Pro CPUID Return Values	8

1.0 INTRODUCTION

As the Intel Architecture evolves, with the addition of new generations and models of processors (8086, 8088, Intel 286, Intel386™, Intel486™, Pentium® processors, and Pentium Pro processors), it is essential for Intel to provide increasingly sophisticated means for software to 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

You can obtain new Intel processor signature and feature bits information from the user's manual, programmer's reference manual or appropriate documentation for a processor. In addition, you can receive 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 provides 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 you 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, and each execution can have a different parameter value in the EAX register. The output depends on the value in the EAX register, as specified in Table 1. If you want the CPUID instruction to determine the highest acceptable value in the EAX register, the program should set the EAX register parameter value to 0. Always use a parameter value that is less than or equal to this highest returned value.

3.1 Vendor-ID String

If the EAX register contains a value of 0, the CPUID instruction returns the vendor identification string 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. So the presence of the **GenuineIntel** string is an assurance that the CPUID instruction and the processor signature are implemented as described in this document.

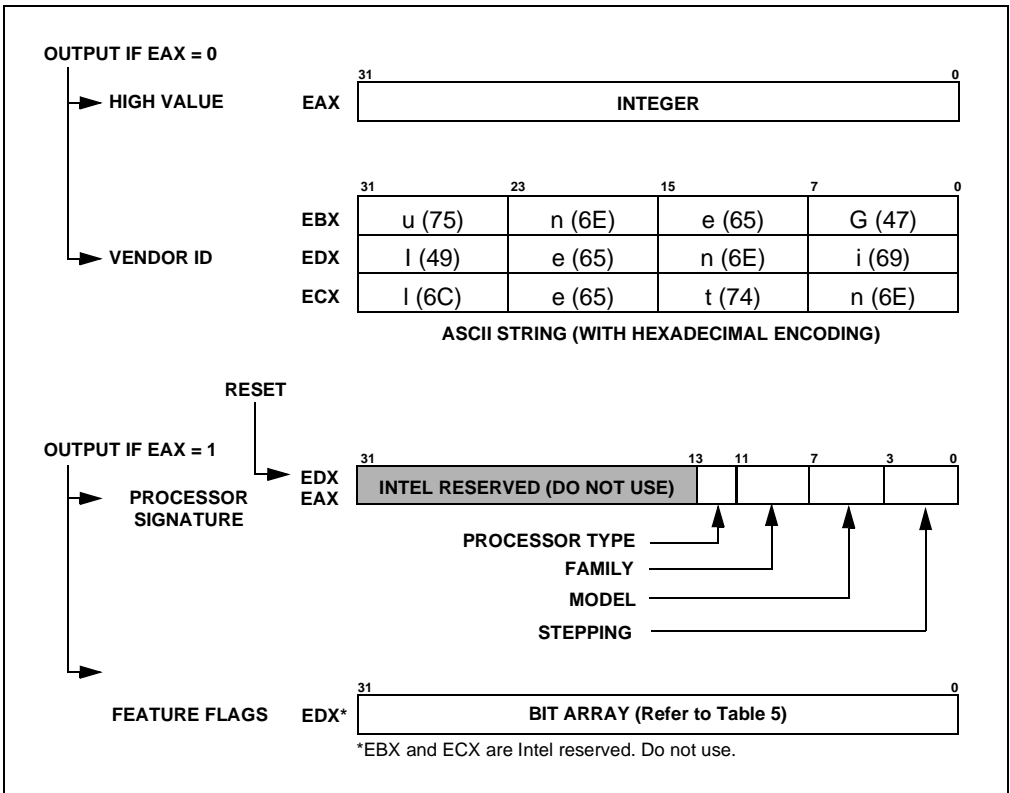


Figure 1. CPUID Instruction Outputs

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.)
EAX = 2	EAX:EBX:ECX:EDX ← Processor configuration parameters
$3 \leq \text{EAX} \leq \text{highest value}$	Intel reserved
EAX > highest value	EAX:EBX:ECX:EDX ← Undefined (Do not use.)

Table 2. Processor Type (Bit Positions 13 and 12)

Value	Description
00	Original OEM processor
01	OverDrive [®] Processor
10	Dual processor
11	Intel reserved. (Do not use.)

3.2 Processor Signature

Beginning with the Intel386 processor family, the processor can return a signature at reset. Processors that implement the CPUID instruction return the processor signature when they either run the CPUID instruction or at reset. Figure 1 shows the format of the signature for the Intel486 and Pentium processor families. Table 3 shows the values currently defined for these processors. (The high-order 18 bits are undefined and reserved.)

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

The family values, specified in bit positions 8 through 11, indicates whether the processor belongs to the Intel386, Intel486, Pentium or Pentium Pro family of processors.

The model number, specified in bits 4 through 7, indicates the processor's family model number, while the stepping number in bits 0 through 3 indicates the revision number of that model.

Older versions of Intel486 SX, Intel486 DX and IntelDX2 processors do not support the CPUID instruction, so they can only return the processor signature at reset. Refer to Table 3 to determine which processors support the CPUID instruction.

Figure 2 shows the format of the processor signature for Intel386 processors. Table 4 shows the values currently defined for Intel386 processors.

Table 3. Intel486, Pentium Processor, and Pentium Pro Processor Signatures

Type	Family	Model	Stepping	Description
00	0100	0000 and 0001	xxxx ¹	Intel486 DX Processors ²
00	0100	0010	xxxx ¹	Intel486 SX Processors ²
00	0100	0011	xxxx ¹	Intel487™ Processors ²
00	0100	0011	xxxx ¹	IntelDX2™ Processors ²
00	0100	0011	xxxx ¹	IntelDX2 OverDrive Processors
00	0100	0100	xxxx ¹	Intel486 SL Processor ²
00	0100	0101	xxxx ¹	IntelSX2™ Processors
00	0100	0111	xxxx ¹	Write-Back Enhanced IntelDX2 Processors
00	0100	1000	xxxx ¹	IntelDX4™ Processors
00, 01	0100	1000	xxxx ¹	IntelDX4 OverDrive Processors
00	0101	0001	xxxx ³	Pentium Processors (510\60, 567\66)
00	0101	0010	xxxx ³	Pentium Processors (735\90, 815\100, 1110\133)
01	0101	0010	xxxx ³	Pentium OverDrive Processor for Pentium Processor (510\60, 567\66)
01	0101	0010	xxxx ³	Pentium OverDrive Processor for Pentium Processor (680\75, 735\90, 815\100, 1000\120, 1100\133)
01	0101	0011	xxxx ³	Pentium OverDrive Processors for Intel486 CPU-based systems
01	0101	0100	xxxx ³	Reserved for a future OverDrive processor for Pentium Processor (680\75, 735\90, 815\100, 1000\120, 1110\133)
01	0101	0101	xxxx ³	Pentium OverDrive Processor for IntelDX4 Processor
00	0110	0001	xxxx ³	Pentium Pro Processor
01	0110	0011	xxxx	Reserved for a future OverDrive Processor for Pentium Pro Processor

1. Intel releases information about minor stepping numbers as needed.
2. This processor does not implement the CPUID instruction.
3. Refer to the Pentium Processor Specifications Update (Order number: 242480), or the Pentium Pro Specifications Update (Order number: 242689) for the latest list of stepping numbers.

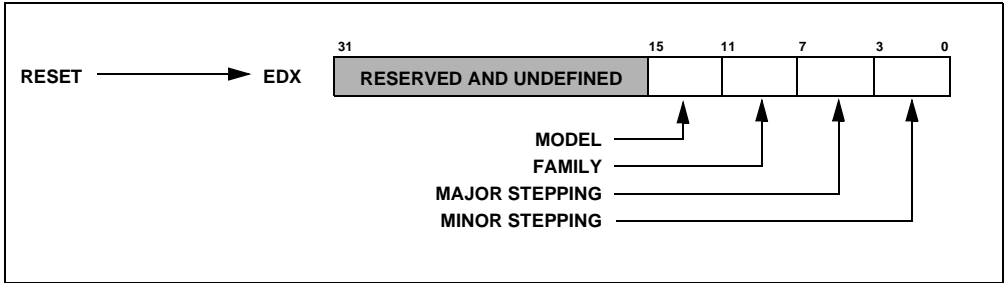


Figure 2. Processor Signature Format on Intel386 Processors

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 [®] Coprocessor

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

3.3 Feature Flags

When the EAX register contains a value of 1, the CPUID instruction loads the EDX register with the feature flags. The feature flags indicate which features the processor supports. However, in future feature flags, a value of one may indicate a feature is not present. Table 5 lists the currently defined feature flag values. For future processors, refer to the program-

mer's reference manual, user's manual, or the appropriate documentation for the latest feature flag values.

Use the feature flags in your applications to determine which processor features are supported. By using the CPUID feature flags to predetermine processor features, your software can detect and avoid incompatibilities.

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	DE	Debugging Extension	The processor supports I/O breakpoints, including the CR4.DE bit for enabling debug extensions and optional trapping of access to the DR4 and DR5 registers.
3	PSE	Page Size Extension	The processor supports 4-Mbyte pages.
4	TSC	Time Stamp Counter	The RDTSC instruction is supported including the CR4.TSD bit for access/privilege control.
5	MSR	Model Specific Registers	Model Specific Registers are implemented with the RDMSR, WRMSR instructions.
6	PAE	Physical Address Extension	Physical addresses greater than 32 bits are supported.
7	MCE	Machine Check Exception	Machine Check Exception, Exception 18, and the CR4.MCE enable bit are supported.
8	CX8	CMPXCHG8 Instruction Supported	The compare and exchange 8 bytes instruction is supported.
9	APIC	Local APIC Supported	The processor contains a local APIC.
10-11		Reserved	(see note) ¹
12	MTRR	Memory Type Range Registers	The Processor supports the Memory Type Range Registers specifically the MTRR_CAP register.
13	PGE	Page Global Enable	The global bit in the PDE's and PTE's and the CR4.PGE enable bit are supported.
14	MCA	Machine Check Architecture	The Machine Check Architecture is supported, specifically the MCG_CAP register.
15	CMOV	Conditional Move Instruction Supported	The processor supports CMOVcc, and if the FPU feature flag (bit 0) is also set, supports the FCMOVcc and FCOMI instructions.
16-31		Reserved	(see note) ¹

1. Processor specific features are outside the scope of this Application Note and are not detailed here. This information is available in the respective processor developer guides, or is available with the appropriate non-disclosure agreement in place. Contact Intel Corporation for details.

3.4 Processor Configuration Information

When the EAX register contains a value of 2, the CPUID instruction loads the EAX, EBX, ECX and EDX registers with descriptors that indicate the processor’s cache characteristics. The lower 8 bits of the EAX register (AL) contain 1. Other values of AL are reserved for future use.

The remainder of the EAX register, and the EBX, ECX, and EDX registers, contain valid 8 bit descriptors. Table 6 shows that valid 8 bit descriptors may be identified as such because their MSB is set to 0. To decode descriptors, move sequentially from the most significant byte of the register down through the least significant byte of the register. Table 7 lists the current descriptor values and their respective cache characteristics. This list will be extended in the future as necessary.

Table 6. Descriptor Formats

Register MSB	Descriptor Type	Description
1	Reserved	Reserved for future use.
0	8 bit descriptors	Descriptors point to a parameter table to identify cache characteristics. The descriptor is null if it has a 0 value.

Table 7. Descriptor Decode Values

Descriptor Value	Cache Description
0x00	null
0x01	instruction TLB, 4K pages, 4-way set associative, 64 entries
0x02	instruction TLB, 4M pages, 4-way set associative, 4 entries
0x03	data TLB, 4K pages, 4-way set associative, 64 entries
0x04	data TLB, 4M pages, 4-way set associative, 8 entries
0x06	instruction cache, 8K, 44-way set associative, 32 byte line size
0x0A	data cache, 8K, 2-way set associative, 32 byte line size
0x41	unified cache, 32 byte cache line, 4-way set associative, 128K
0x42	unified cache, 32 byte cache line, 4-way set associative, 256K
0x43	unified cache, 32 byte cache line, 4-way set associative, 512K

Table 8. Pentium Pro Processor CPUID (EAX=2) Return Values

Register	31	23	15	7	0
EAX	03	02	01	01	
EBX	0	0	0	0	
ECX	0	0	0	0	
EDX	06	04	0A	42	

3.5 Output Example

The initial member of the Pentium Pro processor family returns the values shown in Table 8.

As the value of AL = 1, it is valid to interpret the remainder of the registers according to Table 7. Table 8 also shows that the MSB of the EAX register is 0. This indicates that the upper 8 bits constitute an 8 bit descriptor. The remaining register values in Table 8 show that the Pentium Pro processor has the following cache characteristics:

- A data TLB that maps 4K pages, is 4 way set associative, and has 64 entries.
- An instruction TLB that maps 4M pages, is 4 way set associative, and has 4 entries.
- An instruction TLB that maps 4K pages, is 4 way set associative, and has 64 entries.
- An instruction cache that is 8K, is 4 way set associative, and has a 32 byte line size.
- A data TLB that maps 4M pages, is 4 way set associative, and has 8 entries.
- A data cache that is 8K, is 2 way set associative, and has a 32 byte line size.
- A unified cache that is 256K, is 4 way set associative, and has a 32 byte line size.

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 guidelines are intended to 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 the CPUID instruction. 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 the family value 5 (Pentium processor) means there is a floating-point unit on-chip. Use the feature flags for this determination.
- Do not assume processors with higher family or model numbers have all the features of a processor with a lower family or model number. For example, a processor with a family value of 6 (Pentium Pro processor) does not necessarily have all the features of a processor with a family value of 5.

- 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. The software failed to work correctly when the Intel486 processor used the same opcodes for different instructions. The software should have used the stepping information in the processor signature.
- Do not assume a value of 1 in a feature flag indicates that a given feature is present. For future feature flags, a value of 1 may indicate that the specific feature is not present.
- Test feature flags individually and do not make assumptions about undefined bits. For example, it would be a mistake to test the FPU bit by comparing the feature register to a binary 1 with a compare instruction.
- Do not assume 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. For processors that support the TSC (Time Stamp Counter) functionality, system timers can more directly calibrate the processor core block.
- Processor model-specific registers may differ among processors, including in various models of the Pentium processor. Do not use

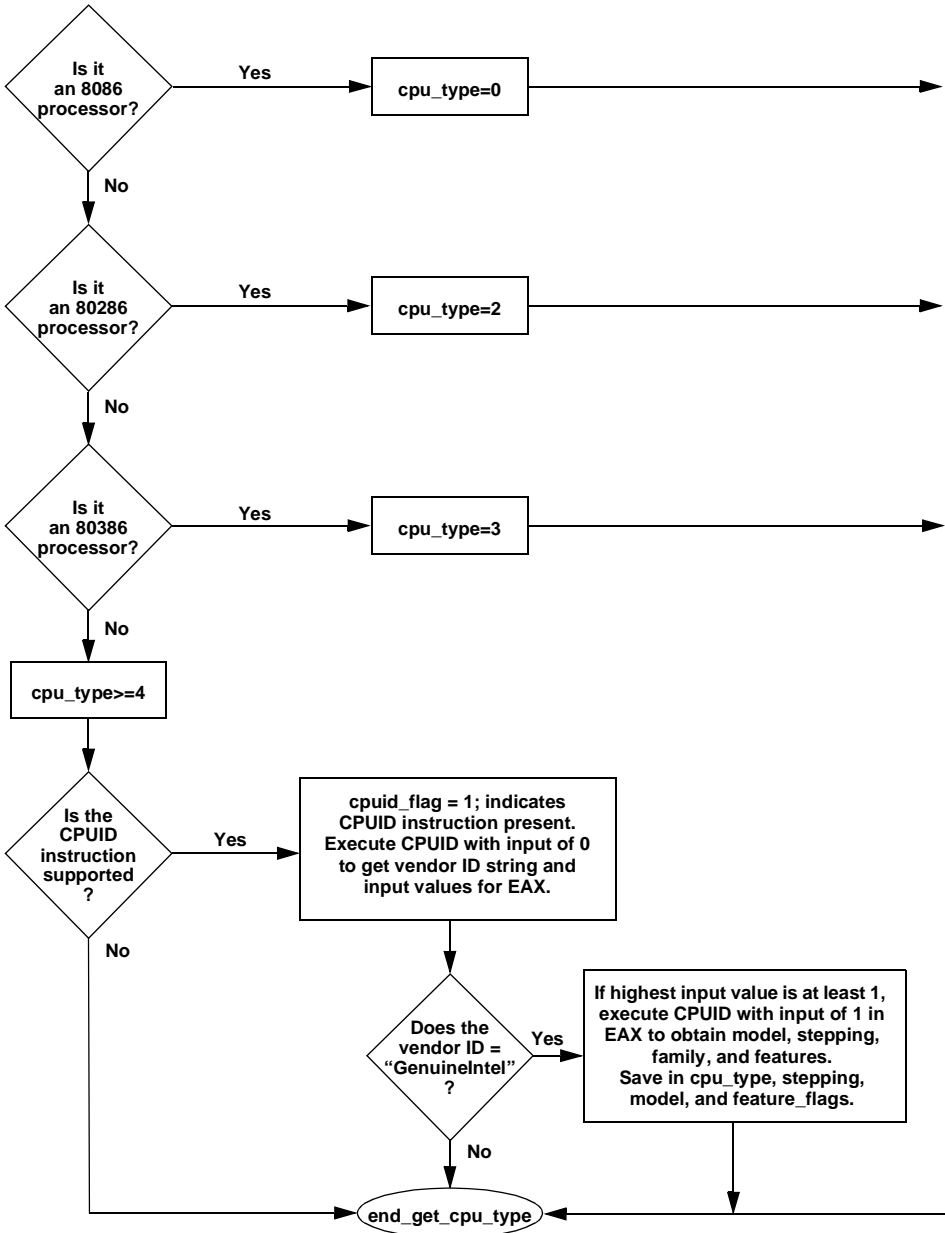
these registers unless identified for the installed processor. This is particularly important for systems upgradeable with an OverDrive processor. The model specific registers in the OverDrive processor are likely to differ from those in the OEM processor. In general, if an OverDrive processor is detected, model specific registers should not be used.

5.0 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. (See Figure 3.) This program example contains the following two procedures:

- **get_cpu_type** identifies the processor type. Figure 4 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, Pentium, and Pentium Pro 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

6.0 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. This code is shown in Example 2 and Example 3. The results, which are displayed on the monitor, identify the installed processor and fea-

tures. 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.

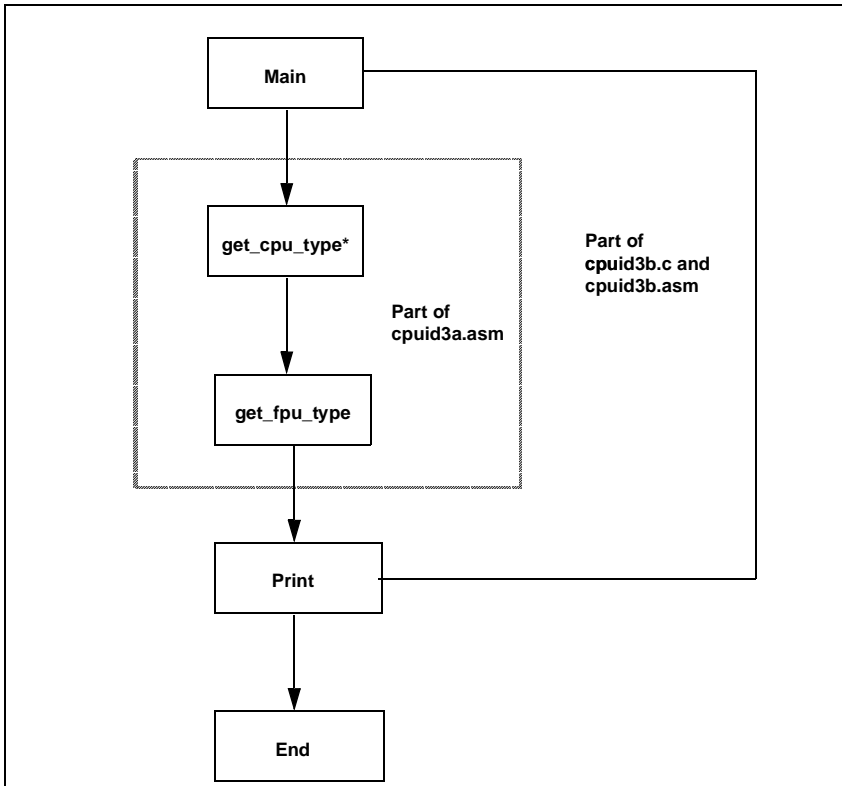


Figure 4. Flow of Processor Identification Extraction Procedures

Example 1. Processor Identification Extraction Procedure

```

;      Filename:  cpuid3a.asm
;      Copyright 1993, 1994, 1995 by Intel Corp.
;
;      This program has been developed by Intel Corporation.  Intel
;      has various intellectual property rights which it may assert
;      under certain circumstances, such as 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 program, 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 program
;      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(R) family processor
;                      6=Pentium(R) Pro 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, Pentium(R), and Pentium(R) Pro
;      processors in the real-address mode.
;
;      To assemble this code with TASM, add the JUMPS directive.
;      jumps                ; Uncomment this line for TASM

      TITLE      cpuid3
      DOSSEG
      .model     small

CPU_ID  MACRO

```

```

        db          0fh          ; Hardcoded CPUID instruction
        db          0a2h

ENDM

        .data
public   _cpu_type
public   _fpu_type
public   _v86_flag
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
_v86_flag      db          0
_cpuid_flag     db          0
_intel_CPU     db          0
_vendor_id     db          "-----"
_intel_CPU     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_typeproc

;
;   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
jne     check_80286     ; go check for 80286
push   sp               ; double check with push sp
pop     dx               ; if value pushed was different
cmp     dx, sp           ; means it's really not an 8086
jne     end_cpu_type    ; jump if processor is 8086/8088
mov     _cpu_type, 10h   ; indicate unknown processor
jmp     end_cpu_type

; 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:
smsw   ax               ; save machine status word
and    ax, 1            ; isolate PE bit of MSW
mov    _v86_flag, al    ; save PE bit to indicate V86

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     ; jump if 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
    CPUID                 ; 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

    cmp                   dword ptr intel_id, ebx
    jne                   end_cpuid_type
    cmp                   dword ptr intel_id[+4], edx
    jne                   end_cpuid_type
    cmp                   dword ptr intel_id[+8], ecx
    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
        mov        _cpu_id, eax      ; 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_typeproc

;       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
    fldl                    ; 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.  Intel
;      has various intellectual property rights which it may assert
;      under certain circumstances, such as 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 program,
;      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
;      program 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, Pentium(R) and
;      Pentium(R) Pro 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
DE_FLAG       equ 0004h
PSE_FLAG      equ 0008h
TSC_FLAG      equ 0010h
MSR_FLAG      equ 0020h
PAE_FLAG      equ 0040h
MCE_FLAG      equ 0080h
CX8_FLAG      equ 0100h
APIC_FLAG     equ 0200h
MTRR_FLAG     equ 1000h
PGE_FLAG      equ 2000h
MCA_FLAG      equ 4000h
CMOV_FLAG     equ 8000h

```

```

.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      "\n 80487SX math coprocessor$"
cp_486sx      db      "\n 80486SX processor$"

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

intel486_msg  db      "\n Genuine Intel486(TM) processor$"
intel486dx_msg db      "\n Genuine Intel486(TM) DX processor$"
intel486sx_msg db      "\n Genuine Intel486(TM) SX processor$"
inteldx2_msg  db      "\n Genuine IntelDX2(TM) processor$"
intelsx2_msg  db      "\n Genuine IntelSX2(TM) processor$"
inteldx4_msg  db      "\n Genuine IntelDX4(TM) processor$"
inteldx2wb_msg db      "\n Genuine Write-Back Enhanced"
              db      "\n IntelDX2(TM) processor$"
pentium_msg   db      "\n Genuine Intel Pentium(R) processor$"
pentiumpro_msg db      "\n Genuine Intel Pentium(R) Pro 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:             $"
stepping_msg  db      13,10,"Stepping:         "
cr_1f        db      13,10,"$"

```

```

turbo_msg      db      13,10,"The processor is an OverDrive(R)"
               db      " upgrade processor$"
dp_msg         db      13,10,"The processor is the upgrade"
               db      "processor in a dual processor system$"
fpu_msg        db      13,10,"The processor contains an on-chip"
               db      "FPU$"
vme_msg        db      13,10,"The processor supports Virtual"
               db      "Mode Extensions$"
de_msg         db      13,10,"The processor supports Debugging"
               db      " Extensions$"
pse_msg        db      13,10,"The processor supports Page Size"
               db      " Extensions$"
tsc_msg        db      13,10,"The processor supports Time Stamp"
               db      " Counter$"
msr_msg        db      13,10,"The processor supports Model"
               db      "Specific Registers$"
pae_msg        db      13,10,"The processor supports Physical"
               db      "Address Extensions$"
mce_msg        db      13,10,"The processor supports Machine"
               db      "Check Exceptions$"
cx8_msg        db      13,10,"The processor supports the"
               db      "CMPXCHG8B instruction$"
apic_msg       db      13,10,"The processor contains an on-chip"
               db      " APIC$"
mtrr_msg       db      13,10,"The processor supports Memory Type"
               db      " Range Registers$"
pge_msg        db      13,10,"The processor supports Page Global"
               db      " Enable$"
mca_msg        db      13,10,"The processor supports Machine"
               db      "Check Architecture$"
cmov_msg       db      13,10,"The processor supports Conditional"
               db      "Move Instruction$"

not_intel      db      "t least an 80486 processor."
               db      13,10,"It does not contain a Genuine"
               db      "Intel part and as a result,"
               db      "the",13,10,"CPUID"
               db      " detection information cannot be"
               db      "determined 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_pentiumpro_type
    mov     dx, offset pentium_msg
    jmp     print_common

print_pentiumpro_type:
    cmp     _cpu_type, 6    ; if 6, print Pentium Pro processor
    jne     print_unknown_type
```

```

        mov     dx, offset pentiumpro_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_VME
        mov     dx, offset fpu_msg
        mov     ah, 9h
        int     21h

```

check_VME:

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

check_DE:

```
mov    ax, word ptr _features_edx
and    ax, DE_FLAG      ; check for DE
jz     check_PSE
mov    dx, offset de_msg
mov    ah, 9h
int    21h
```

check_PSE:

```
mov    ax, word ptr _features_edx
and    ax, PSE_FLAG     ; check for PSE
jz     check_TSC
mov    dx, offset pse_msg
mov    ah, 9h
int    21h
```

check_TSC:

```
mov    ax, word ptr _features_edx
and    ax, TSC_FLAG    ; check for TSC
jz     check_MSR
mov    dx, offset tsc_msg
mov    ah, 9h
int    21h
```

check_MSR:

```
mov    ax, word ptr _features_edx
and    ax, MSR_FLAG    ; check for MSR
jz     check_PAE
mov    dx, offset msr_msg
mov    ah, 9h
int    21h
```

check_PAE:

```
mov    ax, word ptr _features_edx
and    ax, PAE_FLAG    ; check for PAE
jz     check_MCE
mov    dx, offset pae_msg
mov    ah, 9h
int    21h
```

check_MCE:

```

mov     ax, word ptr _features_edx
and     ax, MCE_FLAG      ; check for MCE
jz      check_CX8
mov     dx, offset mce_msg
mov     ah, 9h
int     21h

```

check_CX8:

```

mov     ax, word ptr _features_edx
and     ax, CX8_FLAG      ; check for CMPXCHG8B
jz      check_APIC
mov     dx, offset cx8_msg
mov     ah, 9h
int     21h

```

check_APIC:

```

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

```

check_MTRR:

```

mov     ax, word ptr _features_edx
and     ax, MTRR_FLAG     ; check for MTRR
jz      check_PGE
mov     dx, offset mtrr_msg
mov     ah, 9h
int     21h

```

check_PGE:

```

mov     ax, word ptr _features_edx
and     ax, PGE_FLAG      ; check for PGE
jz      check_MCA
mov     dx, offset pge_msg
mov     ah, 9h
int     21h

```

check_MCA:

```

mov     ax, word ptr _features_edx
and     ax, MCA_FLAG      ; check for MCA
jz      check_CMOV
mov     dx, offset mca_msg
mov     ah, 9h
int     21h

```

```
check_CMOV:
    mov     ax, word ptr _features_edx
    and     ax, CMOV_FLAG    ; check for CMOV
    jz     end_print
    mov     dx, offset cmov_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.  Intel has */
/* various intellectual property rights which it may assert under */
/* certain circumstances, such as 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 program, 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 program 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, Pentium(R), and */
/* Pentium(R) Pro processors in the real-address mode.     */
/*                                                       */

#define FPU_FLAG      0x0001
#define VME_FLAG      0x0002
#define DE_FLAG       0x0004
#define PSE_FLAG      0x0008
#define TSC_FLAG      0x0010
#define MSR_FLAG      0x0020
#define PAE_FLAG      0x0040
#define MCE_FLAG      0x0080
#define CX8_FLAG      0x0100
#define APIC_FLAG     0x0200
#define MTRR_FLAG     0x1000
#define PGE_FLAG      0x2000
#define MCA_FLAG      0x4000
#define CMOV_FLAG     0x8000

```

```
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(R) processor");
else if (cpu_type == 6)
    printf("Genuine Intel Pentium(R) Pro 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(R)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 & VME_FLAG)
        printf("\nThe processor supports Virtual Mode \
Extensions");
        if (features_edx & DE_FLAG)
            printf("\nThe processor supports the Debugging\
Extensions");
            if (features_edx & PSE_FLAG)
                printf("\nThe processor supports Page Size \
Extensions");

```

```
        if (features_edx & TSC_FLAG)
            printf("\nThe processor supports Time Stamp \
Counter");
        if (features_edx & MSR_FLAG)
            printf("\nThe processor supports Model Specific \
Registers");
        if (features_edx & PAE_FLAG)
            printf("\nThe processor supports Physical Address \
Extension");
        if (features_edx & MCE_FLAG)
            printf("\nThe processor supports Machine Check \
Exceptions");
        if (features_edx & CX8_FLAG)
            printf("\nThe processor supports the CMPXCHG8B \
instruction");
        if (features_edx & APIC_FLAG)
            printf("\nThe processor contains an on-chip APIC");
        if (features_edx & MTRR_FLAG)
            printf("\nThe processor supports the Memory Type \
Range Registers");
        if (features_edx & PGE_FLAG)
            printf("\nThe processor supports Page Global
Enable");
        if (features_edx & MCA_FLAG)
            printf("\nThe processor supports the Machine Check \
Architecture");
        if (features_edx & CMOV_FLAG)
            printf("\nThe processor supports the Conditional \
Move Instruction");
    } 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");
}
```