Intel Processor Identification and the CPUID Instruction
Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

The Pentium® and Pentium Pro processors may contain design defects or errors known as errata. Current characterized errata are available on request.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Intel may make changes to specifications and product descriptions at any time, without notice.

*Third-party brands and names are the property of their respective owners.

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

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

COPYRIGHT © INTEL CORPORATION, 1996
CONTENTS

1.0. INTRODUCTION ............................................. 5
   1.1. Update Support ......................................... 5

2.0. DETECTING THE CPUID INSTRUCTION ....... 5

3.0. OUTPUTS OF THE CPUID INSTRUCTION ... 6
   3.1. Vendor-ID String ...................................... 7
   3.2. Processor Signature .................................. 7
   3.3. Feature Flags ......................................... 9

3.4. Cache Size and Format Information .......... 9

3.5. Output Example ....................................... 11

4.0. USAGE GUIDELINES ...................................... 12

5.0. PROPER IDENTIFICATION SEQUENCE .......... 13

6.0. USAGE PROGRAM EXAMPLE ........................... 14
## REVISION HISTORY

<table>
<thead>
<tr>
<th>Revision</th>
<th>Revision History</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>-001</td>
<td>Original Issue.</td>
<td>05/93</td>
</tr>
<tr>
<td>-002</td>
<td>Modified Table 2. Intel486™ and Pentium® Processor Signatures</td>
<td>10/93</td>
</tr>
<tr>
<td>-003</td>
<td>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.</td>
<td>09/94</td>
</tr>
<tr>
<td>-004</td>
<td>Updated with Pentium Pro and OverDrive processors information. Modified Tables1, 3 and 5. Inserted Tables 6, 7 and 8. Inserted Section 3.4. and 3.5.</td>
<td>12/95</td>
</tr>
<tr>
<td>-005</td>
<td>Added Figures 1 and 3. Added Endnotes 1 and 2. Modified Figure 2. Added Assembly code ex in Section 4. Modified Tables 3, 5 and 7. Added two bullets in Section 5.0. Modified cpuid3b.ASM and cpuid3b.C programs to determine if processor features MMX™ technology. Modified Figure 6.0.</td>
<td>11/96</td>
</tr>
</tbody>
</table>
1.0. INTRODUCTION

As the Intel Architecture evolves, with the addition of new generations and models of processors (8086, 8088, Intel286, Intel386™, Intel486™, Pentium® processors, Pentium® OverDrive® processors, Pentium® processors with MMX™ technology, Pentium OverDrive processors with MMX technology, Pentium® Pro processors and P6 family processors with MMX technology), 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 or architectural 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, but only upon reset.
- As the Intel Architecture evolved, Intel extended the processor signature identification into the cpum instruction. The cpum 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 cpum instruction is extensible.

This Application Note explains how to use the cpum instruction in software applications, BIOS implementations, and various processor tools. By taking advantage of the cpum 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

Starting with the Intel486 family and subsequent Intel processors, Intel provides a straightforward method for determining whether the processor's internal architecture is able to execute the CPUID instruction. 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 executable. ¹ See Figure 1.

![Figure 1. Flag Register Evolution](image)

The POPF, POPFD, PUSHF, and PUSHFD instructions are used to access the Flags, Eflags register. The program examples at the end of this Application Note show how you use the PUSHFD instruction to read and...

Footnotes

¹ Only in some Intel486 and succeeding processors. Bit 21 in the Intel386 processor's Eflag register cannot be changed by software, and the Intel386 cannot execute the CPUID instruction. Execution of CPUID on a processor that does not support this instruction will result in an invalid opcode exception.
the POPFD instruction to change the value of the ID flag.

3.0. OUTPUTS OF THE CPUID INSTRUCTION

Figure 2 summarizes the outputs of the CPUID instruction.

The function of the CPUID instruction is fully dependent upon the contents of the EAX register. This means, by placing different values in the EAX register and then executing CPUID, the CPUID instruction will perform a specific function dependent upon whatever value is resident in the EAX register (see Table 1). In order to determine the highest acceptable value for the EAX register input and CPUID operation, the program should set the EAX register parameter value to "0" and then execute the CPUID instruction as follows:

```
MOV EAX, 00H
CPUID
```

After the execution of the CPUID instruction, a return value will be present in the EAX register. Always use a EAX parameter value that is equal to or greater than zero and less than or equal to this highest EAX "returned" value. The values returned by the processor in response to a CPUID instruction with EAX set to a value higher than appropriate for that processor are model specific and should not be relied upon.

![Figure 2. CPUID Instruction Outputs](image-url)
Table 1. Effects of EAX Contents on CPUID Instruction Output

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Outputs of CPUID</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX = 0</td>
<td>EAX ← Highest value recognized by CPUID instruction</td>
</tr>
<tr>
<td></td>
<td>EBX:EDX:ECX ← Vendor identification string</td>
</tr>
<tr>
<td>EAX = 1</td>
<td>EAX ← Processor signature</td>
</tr>
<tr>
<td></td>
<td>EDX ← Feature flags</td>
</tr>
<tr>
<td></td>
<td>EBX:ECX ← Intel reserved (Do not use.)</td>
</tr>
<tr>
<td>EAX = 2</td>
<td>EAX:EBX:ECX:EDX ← Processor configuration parameters</td>
</tr>
<tr>
<td>3 ≤ EAX ≤ highest value</td>
<td>Intel reserved</td>
</tr>
<tr>
<td>EAX &gt; highest value</td>
<td>EAX:EBX:ECX:EDX ← Undefined (Do not use.)</td>
</tr>
</tbody>
</table>

3.1. Vendor-ID String

In addition to returning the highest value in the EAX register, the Intel Vendor-ID string can be simultaneously verified as well. If the EAX register contains an input value of 0, the CPUID instruction also returns the vendor identification string in the EBX, EDX, and ECX registers (see Figure 2). 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. If the "GenuineIntel" string is not returned after execution of the CPUID instruction, do not rely upon the information described in this document to interpret the information returned by the CPUID instruction.

3.2. Processor Signature

Beginning with the Intel486 processor family, the processor will return a processor identification signature value after reset in the EDX register (see Figure 3).

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.

Footnotes

2 All Intel486 SL Enhanced and Write-Back enhanced processors are capable of executing the CPUID instruction. See Table 3.
Table 3. Intel486™, Pentium® Processor Family, OverDrive® and Pentium® Pro Processor Signatures

<table>
<thead>
<tr>
<th>Type</th>
<th>Family</th>
<th>Model</th>
<th>Stepping</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0100</td>
<td>0000 and 0001</td>
<td>xxxx (1)</td>
<td>Intel486™ DX Processors</td>
</tr>
<tr>
<td>00</td>
<td>0100</td>
<td>0010</td>
<td>xxxx (1)</td>
<td>Intel486 SX Processors</td>
</tr>
<tr>
<td>00</td>
<td>0100</td>
<td>0011</td>
<td>xxxx (1)</td>
<td>Intel487 Processors</td>
</tr>
<tr>
<td>00</td>
<td>0100</td>
<td>0011</td>
<td>xxxx (1)</td>
<td>IntelDX2™ Processors</td>
</tr>
<tr>
<td>00</td>
<td>0100</td>
<td>0011</td>
<td>xxxx (1)</td>
<td>IntelDX2 OverDrive® Processors</td>
</tr>
<tr>
<td>00</td>
<td>0100</td>
<td>0100</td>
<td>xxxx (3)</td>
<td>Intel486 SL Processor</td>
</tr>
<tr>
<td>00</td>
<td>0100</td>
<td>0101</td>
<td>xxxx (1)</td>
<td>IntelSX2™ Processors</td>
</tr>
<tr>
<td>00</td>
<td>0100</td>
<td>0111</td>
<td>xxxx (3)</td>
<td>Write-Back Enhanced IntelDX2 Processors</td>
</tr>
<tr>
<td>00</td>
<td>0100</td>
<td>1000</td>
<td>xxxx (3)</td>
<td>IntelDX4™ Processors</td>
</tr>
<tr>
<td>00, 01</td>
<td>0100</td>
<td>1000</td>
<td>xxxx (3)</td>
<td>IntelDX4 OverDrive Processors</td>
</tr>
<tr>
<td>00</td>
<td>0101</td>
<td>0001</td>
<td>xxxx (2)</td>
<td>Pentium® Processors (60, 66)</td>
</tr>
<tr>
<td>00</td>
<td>0101</td>
<td>0010</td>
<td>xxxx (2)</td>
<td>Pentium Processors (75, 90, 100, 120, 133, 150, 166, 200)</td>
</tr>
<tr>
<td>00</td>
<td>0101</td>
<td>0001</td>
<td>xxxx (2)</td>
<td>Pentium OverDrive Processor for Pentium Processor (60, 66)</td>
</tr>
<tr>
<td>00</td>
<td>0101</td>
<td>0010</td>
<td>xxxx (2)</td>
<td>Pentium OverDrive Processor for Pentium Processor (75, 90, 100, 120, 133)</td>
</tr>
<tr>
<td>01</td>
<td>0101</td>
<td>0011</td>
<td>xxxx (2)</td>
<td>Pentium OverDrive Processors for Intel486 CPU-based systems</td>
</tr>
<tr>
<td>00</td>
<td>0101</td>
<td>0100</td>
<td>xxxx (2)</td>
<td>Pentium Processor with MMX™ Technology (166, 200)</td>
</tr>
<tr>
<td>01</td>
<td>0101</td>
<td>0100</td>
<td>xxxx (2)</td>
<td>Reserved for a future OverDrive processor with MMX™ Technology for Pentium Processor (75-200)</td>
</tr>
<tr>
<td>00</td>
<td>0110</td>
<td>0001</td>
<td>xxxx (2)</td>
<td>Pentium Pro Processor</td>
</tr>
<tr>
<td>00</td>
<td>0110</td>
<td>0011</td>
<td>xxxx (2)</td>
<td>P6 Family Processor with MMX Technology</td>
</tr>
<tr>
<td>01</td>
<td>0110</td>
<td>0011</td>
<td>xxxx</td>
<td>Reserved for a future OverDrive Processor for Pentium Pro Processor</td>
</tr>
</tbody>
</table>

NOTES:

1. This processor does not implement the CPUID instruction.
2. Refer to the Intel486® Family documentation, or the Pentium® Processor Specification Update (Order Number 242480), or the Pentium® Pro Processor Specification Update (Order Number 242689) for the latest list of stepping numbers.
3. Stepping 3 implements the CPUID instruction.
Figure 4 shows the format of the processor signature for Intel386 processors, which are different from other processors. Table 4 shows the values currently defined for these Intel386 processors.

### 3.3. Feature Flags

When the EAX register contains a value of 1, the CPUID instruction (in addition to loading the processor signature in the EAX register) loads the EDX register with the feature flags. The current feature flags (when Flag = 1) indicate what features the processor supports. However, in future feature flags, a value of one may indicate a feature has been removed. 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.

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.

### 3.4. Cache Size and Format 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 a value that identifies the number of times the CPUID has to be executed to obtain a complete image of the processor's caching systems. For example, the Pentium Pro processor returns a value of 1 in the lower 8 bits of the EAX register to indicate that the CPUID instruction need only be executed once (with EAX = 2) to obtain a complete image of the processor configuration.

The remainder of the EAX register, and the EBX, ECX, and EDX registers, contain valid 8 bit descriptors. Table 6 shows that a most significant bit of zero indicates a valid 8-bit descriptor. 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 5. Feature Flag Values

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description when Flag = 1</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>FPU</td>
<td>Floating-point unit on-chip</td>
<td>The processor contains an FPU that supports the Intel387 floating-point instruction set.</td>
</tr>
<tr>
<td>1</td>
<td>VME</td>
<td>Virtual Mode Extension</td>
<td>The processor supports extensions to virtual-8086 mode.</td>
</tr>
<tr>
<td>2</td>
<td>DE</td>
<td>Debugging Extension</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>PSE</td>
<td>Page Size Extension</td>
<td>The processor supports 4-Mbyte pages.</td>
</tr>
<tr>
<td>4</td>
<td>TSC</td>
<td>Time Stamp Counter</td>
<td>The RDTSC instruction is supported including the CR4.TSD bit for access/privilege control.</td>
</tr>
<tr>
<td>5</td>
<td>MSR</td>
<td>Model Specific Registers</td>
<td>Model Specific Registers are implemented with the RDMSR, WRMSR instructions.</td>
</tr>
<tr>
<td>6</td>
<td>PAE</td>
<td>Physical Address Extension</td>
<td>Physical addresses greater than 32 bits are supported.</td>
</tr>
<tr>
<td>7</td>
<td>MCE</td>
<td>Machine Check Exception</td>
<td>Machine Check Exception, Exception 18, and the CR4.MCE enable bit are supported.</td>
</tr>
<tr>
<td>8</td>
<td>CX8</td>
<td>CMPXCHG8 Instruction Supported</td>
<td>The compare and exchange 8 bytes instruction is supported.</td>
</tr>
<tr>
<td>9</td>
<td>APIC</td>
<td>On-chip APIC Hardware Supported (1)</td>
<td>The processor contains a local APIC.</td>
</tr>
<tr>
<td>10-11</td>
<td>—</td>
<td>Reserved</td>
<td>Do not count on their value.</td>
</tr>
<tr>
<td>12</td>
<td>MTRRR</td>
<td>Memory Type Range Registers</td>
<td>The Processor supports the Memory Type Range Registers specifically the MTRR_CAP register.</td>
</tr>
<tr>
<td>13</td>
<td>PGE</td>
<td>Page Global Enable</td>
<td>The global bit in the PDEs and PTEs and the CR4.PGE enable bit are supported.</td>
</tr>
<tr>
<td>14</td>
<td>MCA</td>
<td>Machine Check Architecture</td>
<td>The Machine Check Architecture is supported, specifically the MCG_CAP register.</td>
</tr>
<tr>
<td>15</td>
<td>CMOV</td>
<td>Conditional Move Instruction Supported</td>
<td>The processor supports CMOVcc, and if the FPU feature flag (bit 0) is also set, supports the FCMOVcc and FCOMI instructions.</td>
</tr>
<tr>
<td>16-22</td>
<td>—</td>
<td>Reserved</td>
<td>Do not count on their value.</td>
</tr>
<tr>
<td>23</td>
<td>MMX technology</td>
<td>Intel Architecture MMX™ Technology supported</td>
<td>The processor supports the MMX Technology instruction set extensions to Intel Architecture.</td>
</tr>
<tr>
<td>24-31</td>
<td>—</td>
<td>Reserved</td>
<td>Do not count on their value.</td>
</tr>
</tbody>
</table>

NOTES:
1. The processor contains a software-accessible Local APIC.
### Table 6. Descriptor Formats

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

### Table 7. Descriptor Decode Values

<table>
<thead>
<tr>
<th>Descriptor Value</th>
<th>Cache Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>Null</td>
</tr>
<tr>
<td>01h</td>
<td>Instruction TLB, 4 Kbyte pages, 4-way set associative, 32 entries</td>
</tr>
<tr>
<td>02h</td>
<td>Instruction TLB, 4 Mbyte pages, 4-way set associative, 4 entries</td>
</tr>
<tr>
<td>03h</td>
<td>Data TLB, 4 Kbyte pages, 4-way set associative, 64 entries</td>
</tr>
<tr>
<td>04h</td>
<td>Data TLB, 4 Mbyte pages, 4-way set associative, 8 entries</td>
</tr>
<tr>
<td>06h</td>
<td>Instruction cache, 32 byte line size, 4-way set associative, 8 Kbytes</td>
</tr>
<tr>
<td>08h</td>
<td>Instruction cache, 32 byte line size, 4-way set associative, 16 Kbytes</td>
</tr>
<tr>
<td>0Ah</td>
<td>Data cache, 32 byte line size, 2-way set associative, 8 Kbytes</td>
</tr>
<tr>
<td>0Ch</td>
<td>Data cache, 32 byte line size, 2-way set associative, 16 KBytes</td>
</tr>
<tr>
<td>40h</td>
<td>No L2 cache</td>
</tr>
<tr>
<td>41h</td>
<td>Unified cache, 32 byte cache line, 4-way set associative, 128 Kbytes</td>
</tr>
<tr>
<td>42h</td>
<td>Unified cache, 32 byte cache line, 4-way set associative, 256 Kbytes</td>
</tr>
<tr>
<td>43h</td>
<td>Unified cache, 32 byte cache line, 4-way set associative, 512 Kbytes</td>
</tr>
<tr>
<td>44h</td>
<td>Unified cache, 32 byte cache line, 4-way set associative, 1MByte</td>
</tr>
</tbody>
</table>

### 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 32 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.
Table 8. Pentium® Pro Processor, with 256K L2 Cache, CPUID (EAX=2) Example Return Values

<table>
<thead>
<tr>
<th>EAX</th>
<th>31</th>
<th>23</th>
<th>15</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>03h</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBX</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECX</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDX</td>
<td>06h</td>
<td>04h</td>
<td></td>
<td>0Ah</td>
<td>42h</td>
</tr>
</tbody>
</table>

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) may 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 stepplings 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. Only use Model Specific registers that are defined in the BIOS writers guide for that processor.

- Do rely on the result of CPUID algorithm when executed in virtual 8086 mode.

- Do not assume any ordering of stepping numbers. They are assigned arbitrarily.
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 5.) This program example contains the following two procedures:

- **get_cpu_type** identifies the processor type. Figure 5 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 processor, Pentium processor with MMX Technology, OverDrive processor with MMX Technology, Pentium Pro processors and Pentium Pro processors with MMX Technology. 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 5. Flow of Processor get_cpu_type Procedure**

- **cpuid_flag**: Indicates CPUID instruction present. Execute CPUID with input of 0 to get vendor ID string and input values for EAX.
- **Does the vendor ID = "GenuineIntel"?**: If highest input value is at least 1, execute CPUID with input of 1 in EAX to obtain model, stepping, family, and features. Save in cpu_type, stepping, model, and feature_flags.
6.0. USAGE PROGRAM EXAMPLE

The cpuid3b.asm or 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 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 6 presents an overview of the relationship between the three program examples.

Figure 6. Flow of Processor Identification Extraction Procedures
Example 1. Processor Identification Extraction Procedure

; Filename: cpuid3a.asm

; 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 = Intel 386(TM) family processor
4 = Intel 486(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 only.

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
db 0
public _cpuid_flag
db 0
public _intel_CPU
db 0
public _vendor_id
db "--------------"
public _cpu_signature
dd 0
public _features_ecx
dd 0
public _features_edx
dd 0
public _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
pop ax
mov cx, ax
and ax, 0ffffh
push ax
popf
pushf
pop ax
and ax, 0f000h
mov _cpu_type, 0
jne check_80286
push sp
pop dx
mov dx, sp
Intel 286 processor check

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

.check_80286:
    smsw ax
    and ax, 1
    mov _v86_flag, al
    or cx, 0f000h
    push cx
    popf
    pushf
    pop ax
    and ax, 0f000h
    pushf
    jmp _cpu_type, 2
    jz end_cpu_type

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.

.check_80386:
    pushfd
    pop eax
    mov ecx, eax
    xor eax, 40000h
    push eax
    popf
    pushf
    pushd
    pop eax
    xor eax, ecx
    mov _cpu_type, 2
    jz end_cpu_type
    push ecx
    popfd

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.

.check_80486:
    mov _cpu_type, 4
    mov eax, ecx
    xor eax, 200000h
    push eax
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 not 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

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

mov _intel_CPU, 1 ; indicate an Intel processor
cmp eax, 1 ; make sure 1 is valid input for CPUID
jl end_cpuid_type
mov eax, 1 ; if not, jump to end
CPU_ID

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
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.

```assembly
fninit
    mov fp_status, 5a5ah
    fnstsw fp_status
    mov ax, fp_status
    cmp ax, 0
    mov _fpu_type, 0
    jne end_fpu_type

check_control_word:
    fnstcw fp_status
    mov ax, fp_status
    and ax, 103fh
    cmp ax, 3fh
    mov _fpu_type, 0
    jne end_fpu_type
    mov _fpu_type, 1

check infinity:
    cmp _cpu_type, 3
    jne end_fpu_type
    fld1
    fldz
    fdiv
    fchs
    fcompp
    fstsw fp_status
    mov ax, fp_status
    mov _fpu_type, 2
    sahf
    jz end_fpu_type
    mov _fpu_type, 3

end_fpu_type:
    ret

_get_fpu_type endp
```

80287/80387 check for the Intel386 processor

See if infinities matched and jump if 8087 or Intel287 is present; store Intel387 NDP for FPU type
Example 2. Processor Identification Procedure in Assembly Language

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.

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
        mov  es, ax
        mov  sp, not 3
        call _get_cpu_type
        call _get_fpu_type
        call print
        mov  ax, 4c00h
        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
MMX_FLAG equ 800000h

.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$"
cp_486   db   " n 80486DX, 80486DX2 processor or"
cp_486sx db   " 80487SX math coprocessor$"
cp_8087  db   " and an 8087 math coprocessor$"
cp_287   db   " and an 80287 math coprocessor$"
fp_387     db     " and an 80387 math coprocessor$"
intel486_msg     db     " Genuine Intel486(TM) processor$"
intel486dx_msg    db     " Genuine Intel486(TM) DX processor$"
intel486sx_msg    db     " Genuine Intel486(TM) SX processor$"
ineldx2_msg      db     " Genuine IntelDX2(TM) processor$"
inelsx2_msg      db     " Genuine IntelSX2(TM) processor$"
ineldx4_msg      db     " Genuine IntelDX4(TM) processor$"
ineldx2wb_msg    db     " Genuine Write-Back Enhanced$"
inteldx4_msg     db     " IntelDX2(TM) processor$"      
pentium_msg       db     " Genuine Intel Pentium(R) processor$"
pentiumpro_msg   db     " 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 ineldx2_msg
intel_486_4     dw     offset intel486_msg
intel_486_5     dw     offset inelsx2_msg
intel_486_6     dw     offset intel486_msg
intel_486_7     dw     offset ineldx2wb_msg
intel_486_8     dw     offset ineldx4_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: $"

vfpu$         db     13,10,"$"
turbo_msg     db     13,10,"The processor is an OverDrive(R)$"          
dp_msg        db     13,10,"The processor is the upgrade$"
dpup$         db     13,10,"processor in a dual processor system$"

fpu_msg       db     13,10,"The processor contains an on-chip$"

vme_msg       db     13,10,"The processor supports Virtual$"

mode$         db     13,10,"Mode Extensions$"
de_msg        db     13,10,"The processor supports Debugging$"
deext$        db     13,10,"Extensions$"
pse_msg       db     13,10,"The processor supports Page Size$"
pseext$       db     13,10,"Extensions$"
tsc_msg       db     13,10,"The processor supports Time Stamp$"
tscext$       db     13,10,"Counter$"
msr_msg       db     13,10,"The processor supports Model$"
msr$          db     13,10,"Specific Registers$"
pae_msg       db     13,10,"The processor supports Physical$"
paext$        db     13,10,"Address Extensions$"
mce_msg       db     13,10,"The processor supports Machine$"
mce$          db     13,10,"Check Exceptions$"

22
cx8_msg  db  13,10,"The processor supports the"
db  "CMPXCHG8B instruction$"
apic_msg db  13,10,"The processor contains an on-chip"
db  "APICS$"
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$"
cmov_msg db  13,10,"The processor supports Conditional"
db  "Move Instruction$"
mmx_msg db  13,10,"The processor supports Intel Architecture"
db  "MMX(TM) technology$"

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

.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 _cpu_type, 1 ; if set to 1, processor
    je print_cpuid_data ; supports CPUID instruction
    print_86: ; print detailed CPUID info
        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
mov dx, offset cp_486sx
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:
; check for genuine Intel processor

print_486_type:
cmp _cpu_type, 4
jne print_pentium_type
mov ax, word ptr _cpu_signature
shr ax, 4
and ax, 0fh
mov dx, intel_486_0[eax*2]
jmp print_common

print_pentium_type:
cmp _cpu_type, 5
jne print_pentiumpro_type
mov dx, offset pentium_msg
jmp print_common

print_pentiumpro_type:
cmp _cpu_type, 6
jne print_unknown_type
mov dx, offset pentiumpro_msg
jmp print_common

print_unknown_type:
mov dx, offset unknown_msg

print_common:
mov ah, 9h
int 21h

; print family, model, and stepping

print_family:
mov al, _cpu_type
ASC_MSG family_msg

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

print_stepping:
mov ax, word ptr _cpu_signature
and al, 0fh
ASC_MSG stepping_msg

print_upgrade:
mov ax, word ptr _cpu_signature
test ax, 1000h
jz check_dp
mov dx, offset turbo_msg
mov ah, 9h
int 21h

; check for turbo upgrade
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     check_mmx
mov    dx, offset cmov_msg
mov    ah, 9h
int    21h

Check/MMX
mov    Eax, word ptr _features_esdx
and    Eax, MMX_FLAG                ; check for MMX technology
jz     endprint
mov    dx, offset mmx_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 Ox0001
#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
#define MMX_FLAG 0x800000

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 \
30487SX 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 \\nIntelDX2(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", cpu_signature&0xf);
if (cpu_signature & 0x1000)
    printf("nThe processor is an OverDrive(R)upgrade \\nprocessor");
else if (cpu_signature & 0x2000)
    printf("nThe processor is the upgrade processor \\n 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 \n Extensions");
if (features_edx & DE_FLAG)
    printf("nThe processor supports the Debugging\n Extensions");
if (features_edx & PSE_FLAG)
    printf("nThe processor supports Page Size \n Extensions");
if (features_edx & TSC_FLAG)
    printf("nThe processor supports Time Stamp \n Counter");
if (features_edx & MSR_FLAG)
    printf("nThe processor supports Model Specific \n Registers");
if (features_edx & PAE_FLAG)
    printf("nThe processor supports Physical Address \n Extension");
if (features_edx & MCE_FLAG)
    printf("nThe processor supports Machine Check \n Exceptions");
if (features_edx & CX8_FLAG)
    printf("nThe processor supports the CMPXCHG8B \n 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 \n Range Registers");
if (features_edx & PGE_FLAG)
printf("The processor supports Page Global Enable");
if (features_edx & MCA_FLAG)
    printf("The processor supports the Machine Check \nArchitecture");
if (features_edx & CMOV_FLAG)
    printf("The processor supports the Conditional \nMove Instruction");
if (features_edx & MMX_FLAG)
    printf("The processor supports Intel Architecture \nMMX Technology");
else {
    printf("at least an 80486 processor. It does not \ncontain a Genuine Intel part and as a result, the \nCPUID detection \ninformation cannot be determined at this time.");
}
printf("n");
<table>
<thead>
<tr>
<th>State</th>
<th>Sales Offices</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ARIZONA</strong></td>
<td>Intel Corp. 1100 T.G Lee Boulevard Suite 150 Orlando 32822 Tel: (407) 856-8686 FAX: (407) 856-8620</td>
</tr>
<tr>
<td><strong>CALIFORNIA</strong></td>
<td>Intel Corp. 380 North Martingale Road Suite 400 Schaumburg 60173 Tel: (847) 880-0276 FAX: (847) 880-0276</td>
</tr>
<tr>
<td><strong>COLORADO</strong></td>
<td>Intel Corp. 5000 Quorum Drive Suite 750 Dallas 75240 Tel: (214) 233-1325</td>
</tr>
<tr>
<td><strong>CONNECTICUT</strong></td>
<td>Intel Corp. 3550 W. Main Street Suite 236 Boise 83702 Tel: (208) 331-2295</td>
</tr>
<tr>
<td><strong>FLORIDA</strong></td>
<td>Intel Corp. 404 North Executive Drive Suite 401 Garland 75040 Tel: (972) 441-5258</td>
</tr>
<tr>
<td><strong>GEORGIA</strong></td>
<td>Intel Corp. 850 Cross Keys Office Park Fulton 30047 Tel: (706) 628-8686 FAX: (706) 628-8686</td>
</tr>
<tr>
<td><strong>IDAHO</strong></td>
<td>Intel Corp. 2301 Park Center Drive Suite 220 Dayton 45414 Tel: (937) 890-8598</td>
</tr>
<tr>
<td><strong>ILLINOIS</strong></td>
<td>Intel Corp. 3500 West 80th Street Suite 350 Bloomington 54311 Tel: (608) 576-2667 FAX: (608) 576-2667</td>
</tr>
<tr>
<td><strong>INDIANA</strong></td>
<td>Intel Corp. 15254 NW Greenbrier Pkwy Building B Beaverton 97006 Tel: (503) 645-8181</td>
</tr>
<tr>
<td><strong>MARYLAND</strong></td>
<td>Intel Corp. 1800 East 156th Avenue SE Suite 105 Bellevue 98007 Tel: (425) 898-8688</td>
</tr>
<tr>
<td><strong>MASSACHUSETTS</strong></td>
<td>Intel Corp. 3500 Parkline Road Suite 4 Columbia 29223 Tel: (803) 768-7999</td>
</tr>
<tr>
<td><strong>MICHIGAN</strong></td>
<td>Intel Corp. 3550 S. Xanadu Dr. Suite 350 Bloomington 54311 Tel: (608) 576-2667 FAX: (608) 576-2667</td>
</tr>
<tr>
<td><strong>MINNESOTA</strong></td>
<td>Intel Corp. 404 North Executive Drive Suite 401 Garland 75040 Tel: (972) 441-5258</td>
</tr>
<tr>
<td><strong>MISSOURI</strong></td>
<td>Intel Corp. 2950 Expressway Drive Islandia 11723 Tel: (631) 348-7939</td>
</tr>
<tr>
<td><strong>NEVADA</strong></td>
<td>Intel Corp. 561 North Broadway Suite 115 Oklahoma City 73102 Tel: (405) 526-4010</td>
</tr>
<tr>
<td><strong>NEW HAMPSHIRE</strong></td>
<td>Intel Corp. 1500 E. Grand Avenue Suite 150 El Segundo, CA 90245 Tel: (310) 460-7133</td>
</tr>
<tr>
<td><strong>NEW JERSEY</strong></td>
<td>Intel Corp. 2950 Xanadu Drive Suite 401 Columbia 29223 Tel: (803) 768-7999</td>
</tr>
<tr>
<td><strong>NEW MEXICO</strong></td>
<td>Intel Corp. 2950 Expressway Drive Islandia 11723 Tel: (631) 348-7939</td>
</tr>
<tr>
<td><strong>NEW YORK</strong></td>
<td>Intel Corp. 850 Cross Keys Office Park Fulton 14450 Tel: (800) 828-8686 FAX: (718) 233-2651</td>
</tr>
<tr>
<td><strong>OHIO</strong></td>
<td>Intel Corp. 1500 E. Grand Avenue Suite 150 El Segundo, CA 90245 Tel: (310) 460-7133</td>
</tr>
<tr>
<td><strong>OKLAHOMA</strong></td>
<td>Intel Corp. 2950 Expressway Drive Islandia 11723 Tel: (631) 348-7939</td>
</tr>
<tr>
<td><strong>OREGON</strong></td>
<td>Intel Corp. 2950 Expressway Drive Islandia 11723 Tel: (631) 348-7939</td>
</tr>
<tr>
<td><strong>PENNSYLVANIA</strong></td>
<td>Intel Corp. 9500 Quorum Drive Suite 750 Dallas 75240 Tel: (214) 233-1325</td>
</tr>
<tr>
<td><strong>TEXAS</strong></td>
<td>Intel Corp. 2950 Expressway Drive Islandia 11723 Tel: (631) 348-7939</td>
</tr>
<tr>
<td><strong>UTAH</strong></td>
<td>Intel Corp. 9500 Quorum Drive Suite 750 Dallas 75240 Tel: (214) 233-1325</td>
</tr>
<tr>
<td><strong>VERMONT</strong></td>
<td>Intel Corp. 9500 Quorum Drive Suite 750 Dallas 75240 Tel: (214) 233-1325</td>
</tr>
<tr>
<td><strong>WASHINGTON</strong></td>
<td>Intel Corp. 2950 Expressway Drive Islandia 11723 Tel: (631) 348-7939</td>
</tr>
<tr>
<td><strong>WISCONSIN</strong></td>
<td>Intel Corp. 2950 Expressway Drive Islandia 11723 Tel: (631) 348-7939</td>
</tr>
<tr>
<td><strong>WYOMING</strong></td>
<td>Intel Corp. 2950 Expressway Drive Islandia 11723 Tel: (631) 348-7939</td>
</tr>
</tbody>
</table>
# NORTH AMERICAN SERVICE OFFICES

**Computervision**  
*Intel Corporation's North American Preferred Service Provider*

Central Dispatch: 1-800-876-SERV (1-800-876-7378)

<table>
<thead>
<tr>
<th>ALABAMA</th>
<th>GEORGIA</th>
<th>MICHIGAN</th>
<th>NORTH CAROLINA</th>
<th>UTAH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Birmingham</td>
<td>Atlanta</td>
<td>Detroit</td>
<td>Asheville</td>
<td>Salt Lake City</td>
</tr>
<tr>
<td>Huntsville</td>
<td>Honolulu</td>
<td>Flint</td>
<td>Charlotte</td>
<td>White River Junction</td>
</tr>
<tr>
<td>ALASKA</td>
<td>ILLINOIS</td>
<td>Grand Rapids</td>
<td>Greensboro</td>
<td>Virginia</td>
</tr>
<tr>
<td>Anchorage</td>
<td>Chicago</td>
<td>Lansing</td>
<td>Raleigh</td>
<td>Charlottesville</td>
</tr>
<tr>
<td>ARIZONA</td>
<td>Wood Dale</td>
<td>Troy</td>
<td>Wilmington</td>
<td>Richmond</td>
</tr>
<tr>
<td>Phoenix</td>
<td></td>
<td></td>
<td></td>
<td>Roanoke</td>
</tr>
<tr>
<td>Tucson</td>
<td></td>
<td></td>
<td></td>
<td>Virginia Beach</td>
</tr>
<tr>
<td>ARKANSAS</td>
<td></td>
<td></td>
<td></td>
<td>WASHINGTON</td>
</tr>
<tr>
<td>North Little Rock</td>
<td></td>
<td></td>
<td></td>
<td>Renton</td>
</tr>
<tr>
<td>ALASKA</td>
<td></td>
<td></td>
<td></td>
<td>Richland</td>
</tr>
<tr>
<td>CALIFORNIA</td>
<td></td>
<td></td>
<td></td>
<td>WASHINGTON D.C.*</td>
</tr>
<tr>
<td>Concord</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Los Angeles</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ontario</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orange</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Redwood City</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sacramento</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>San Diego</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>San Francisco</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Van Nuys</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COLORADO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colorado Springs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Denver</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONNECTICUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E. Windsor</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Middlebury</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLORIDA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ft. Lauderdale</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jacksonville</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miami</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orlando</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pensacola</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tampa</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GEORGIA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Atlanta</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HAWAII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Honolulu</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILLINOIS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chicago</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wood Dale</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDIANA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carmel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evansville</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ft. Wayne</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>South Bend</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOWA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cedar Rapids</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Davenport</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>West Des Moines</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KANSAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kansas City</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wichita</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KENTUCKY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Louisville</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Madisonville</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOUISIANA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baton Rouge</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>New Orleans</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAINE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Auburn</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARYLAND</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baltimore</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MASSACHUSETTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bedford</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S. Easton</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MICHIGAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detroit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flint</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grand Rapids</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lansing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Troy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MINNESOTA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minneapolis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MISSOURI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Springfield</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Street Louis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MISSISSIPPI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jackson</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW HAMPSHIRE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manchester*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MONTANA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Butte</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEBRASKA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Omaha</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW JERSEY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cherry Hill</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hamilton Township</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Westfield</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW MEXICO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Albuquerque</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW YORK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Albany</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binghampton</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffalo</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Farmingdale</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>New York City</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rochester</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dryden</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NORTH CAROLINA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asheville</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charlotte</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greensboro</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Raleigh</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wilmington</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OHIO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cincinnati</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cleveland</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Columbus</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dayton</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OKLAHOMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oklahoma City</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OREGON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beaverton</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PENNSYLVANIA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camp Hill</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Erie</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pittsburgh</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wayne</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOUTH CAROLINA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charleston</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Columbia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greenville</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TENNESSEE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chattanooga</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Knoxville</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memphis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nashville</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXAS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Austin</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Houston</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dallas</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tyler</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>