Intel Processor Identification and the CPUID Instruction

January 1999

INTEL CONFIDENTIAL
(until publication date)
CONTENTS

<table>
<thead>
<tr>
<th></th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>INTRODUCTION</td>
</tr>
<tr>
<td>1.1</td>
<td>Update Support</td>
</tr>
<tr>
<td>2.0</td>
<td>DETECTING THE CPUID INSTRUCTION</td>
</tr>
<tr>
<td>3.0</td>
<td>OUTPUT OF THE CPUID INSTRUCTION</td>
</tr>
<tr>
<td>3.1</td>
<td>Vendor ID String</td>
</tr>
<tr>
<td>3.2</td>
<td>Processor Signature</td>
</tr>
<tr>
<td>3.3</td>
<td>Feature Flags</td>
</tr>
<tr>
<td>3.4</td>
<td>Cache Size and Format Information</td>
</tr>
<tr>
<td>3.5</td>
<td>SYSENTER/SYSEXIT – SEP Features Bit</td>
</tr>
<tr>
<td>3.6</td>
<td>Pentium® Pro Processor Output Example</td>
</tr>
<tr>
<td>3.7</td>
<td>Pentium® II Processor, model 3 Output Example</td>
</tr>
<tr>
<td>4.0</td>
<td>PROCESSOR SERIAL NUMBER</td>
</tr>
<tr>
<td>4.1</td>
<td>Presence of Processor Serial Number</td>
</tr>
<tr>
<td>4.2</td>
<td>Forming the 96-bit Processor Serial Number</td>
</tr>
<tr>
<td>5.0</td>
<td>USAGE GUIDELINES</td>
</tr>
<tr>
<td>6.0</td>
<td>PROPER IDENTIFICATION SEQUENCE</td>
</tr>
<tr>
<td>7.0</td>
<td>USAGE PROGRAM EXAMPLES</td>
</tr>
<tr>
<td>Revision</td>
<td>Revision History</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-001</td>
<td>Original Issue.</td>
</tr>
<tr>
<td>-002</td>
<td>Modified Table 2, Intel486™ and Pentium® Processor Signatures.</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>
</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 Sections 3.4 and 3.5.</td>
</tr>
<tr>
<td>-005</td>
<td>Added Figures 1 and 3. Added Footnotes 1 and 2. Modified Figure 2. Added Assembly code example 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>
</tr>
<tr>
<td>-006</td>
<td>Modified Table 3. Added reserved for future member of P6 family of processors entry. Modified table header to reflect Pentium® II processor family. Modified Table 5. Added SEP bit definition. Added Section 3.5. Added Section 3.7 and Table 9. Corrected references of P6 family to reflect correct usage. Modified cpuid3a.asm, cpuid3b.asm and cpuid3c.c example code sections to check for SEP feature bit and to check for, and identify, the Pentium® II processor. Added additional disclaimer related to designers and errata.</td>
</tr>
<tr>
<td>-007</td>
<td>Modified Table 2. Added Pentium® II processor, model 5 entry, Modified existing Pentium II processor entry to read “Pentium II processor, model 3”, Modified Table 5. Added additional feature bits, PAT and FXSR, Modified Table 7. Added entries 44h and 45h. Removed the note “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” in section 4.0. Modified cpuid3b.asm and cpuid3c.c example code section to check for, and identify, the Pentium II processor, model 5. Modified existing Pentium II processor code to print Pentium II processor, model 3.</td>
</tr>
<tr>
<td>-008</td>
<td>Added note to identify Intel® Celeron™ processor, model 5 in section 3.2. Modified Table 2. Added Intel Celeron processor &amp; Pentium® OverDrive® processor with MMX™ technology entry. Modified Table 5. Added additional feature bit, PSE-36. Modified cpuid3b.asm and cpuid3c.c example code to check for, and identify, the Intel Celeron processor.</td>
</tr>
<tr>
<td>-009</td>
<td>Added note to identify Pentium® II Xeon™ processor in section 3.2. Modified Table 2. Added Pentium II Xeon processor entry. Modified cpuid3b.asm and cpuid3c.c example code to check for, and identify, the Pentium II Xeon processor.</td>
</tr>
<tr>
<td>-010</td>
<td>No Changes</td>
</tr>
<tr>
<td>-011</td>
<td>Modified Table 2. Added Intel® Celeron® processor, model 6 entry. Modified cpuid3b.asm and cpuid3.c example code to check for, and identify, the Intel Celeron processor, model 6.</td>
</tr>
<tr>
<td>-012</td>
<td>Modified Figure 1 to add the reserved information for the Intel386™ processors. Modified Figure 2. Added the Processor serial number information returned when the CPUID instruction is executed with EAX=3. Modified Table 1. Added the Pentium® III processor and Pentium® III Xeon™ processor. Added Section 4 “Processor serial number”. Modified cpuid3a.asm, cpuid3b.asm and cpuid3.c example code to check for and identify, the Pentium III processor and the Pentium III Xeon processor.</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, Pentium II processors, Pentium II Xeon™ processors, Pentium II Overdrive® processors, Intel Celeron™ processors, Pentium® III processors, and Pentium III Xeon™ processors), it is essential that Intel provide 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:

1. Originally, Intel published code sequences that could detect minor implementation or architectural differences to identify processor generations.
2. 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.
3. 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.

The Pentium III processors, and Pentium III Xeon processors extend the concept of processor identification with the addition of processor serial number. Processor serial number is a 96-bit number accessible through the CPUID instruction. Processor serial number can be used by applications to identify a processor, and by extensions, its system.

This application note explains how to use the CPUID instruction in software applications, BIOS implementations, and various processor 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 developer’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, or visit Intel’s website at http://developer.intel.com/.

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.

The POPF, POPFD, PUSHF, and PUSHFD instructions are used to access the Flags in Eflags register. 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. OUTPUT 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

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 processor cannot execute the CPUID instruction. Execution of CPUID on a processor that does not support this instruction will result in an invalid opcode exception.
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
```

### 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.

### 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, or Upper 32 bits of 96-bit processor serial number</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>EAX = 3</td>
<td>EDX:ECX ← lower 64-bits of 96-bit processor serial number</td>
</tr>
<tr>
<td>4 ≤ 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>

### Figure 1. Flag Register Evolution

After the execution of the CPUID instruction, a return value will be present in the EAX register. Always use an EAX parameter value that is equal to or greater than zero and less than or equal to this highest EAX “returned” value. On current and future IA-32 processors, bit 31 in the EAX register will be clear when CPUID is called with an input parameter greater than the highest value. All other bit 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
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).

<table>
<thead>
<tr>
<th>EDX</th>
<th>Reserved</th>
<th>Type</th>
<th>Family</th>
<th>Model</th>
<th>Stepping</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. EDX Register Value after RESET

Processors that implement the CPUID instruction also return the processor identification signature after reset; however, the CPUID instruction gives you the flexibility of checking the processor signature at any time. Figure 3 shows the format of the signature for the Intel486, Pentium®, Pentium Pro, Pentium II processors, Pentium II Xeon™ processors, Pentium II Overdrive® processors, Intel® Celeron™ processors, Pentium III processors, and Pentium III Xeon processors. Note that the EDX processor signature value after reset is equivalent to the processor signature output value in the EAX register in Figure 2. Table 3 shows the values returned in the EAX register currently defined for these processors. (If the processor serial number is supported the high-order 18 bits are 0. Otherwise the high-order 18 bits are undefined and reserved.)

The processor type, specified in bit positions 12 and 13 of Table 2, 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 P6 family of processors. P6 family processors include all processors based on the Pentium® Pro processor architecture and have a family code equal to 6.

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

The Pentium II processor, model 5, the Pentium II Xeon processor and the Intel Celeron processor, model 5 share the same family and model number. To differentiate between the processors, software should check the cache descriptor values through executing CPUID instruction with EAX = 2. If no L2 cache is returned, the processor is identified as an Intel Celeron processor, model 5. If 1M or 2M L2 cache size is reported, the processor is the Pentium II Xeon processor otherwise it is a Pentium II processor, model 5 or a Pentium II Xeon processor with 512K L2 cache.

The Pentium III processor and the Pentium III Xeon processor share the same family and model number. To differentiate between the processors, software should check the cache descriptor values through executing CPUID instruction with EAX = 2. If 1M or 2M L2 cache size is reported, the processor is the Pentium III Xeon processor otherwise it is a Pentium III processor or a Pentium III Xeon processor with 512K L2 cache.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Original OEM processor</td>
</tr>
<tr>
<td>01</td>
<td>OverDrive® processor</td>
</tr>
<tr>
<td>10</td>
<td>Dual processor</td>
</tr>
<tr>
<td>11</td>
<td>Intel reserved (Do not use.)</td>
</tr>
</tbody>
</table>

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 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.

Footnotes

2 All Intel486 SL-enhanced and Write-Back enhanced processors are capable of executing the CPUID instruction. See Table 3.
Table 3. Intel®486™, Pentium® Processor Family, OverDrive®, Pentium Pro Processor, Pentium II Processor, Pentium II Xeon™ Processor, Pentium II OverDrive® Processor, Intel® Celeron™ Processor, Pentium III Processor, and Pentium III Xeon 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>0010</td>
<td>0000 and 0001</td>
<td>xxx (1)</td>
<td>Intel®486™ DX processors</td>
</tr>
<tr>
<td>00</td>
<td>0010</td>
<td>0010</td>
<td>xxx (1)</td>
<td>Intel®486™ SX processors</td>
</tr>
<tr>
<td>00</td>
<td>0010</td>
<td>0011</td>
<td>xxx (1)</td>
<td>Intel®487™ processors</td>
</tr>
<tr>
<td>00</td>
<td>0010</td>
<td>0011</td>
<td>xxx (1)</td>
<td>Intel®DX2™ processors</td>
</tr>
<tr>
<td>00</td>
<td>0010</td>
<td>0011</td>
<td>xxx (1)</td>
<td>Intel®DX2™ OverDrive® processors</td>
</tr>
<tr>
<td>00</td>
<td>0010</td>
<td>0100</td>
<td>xxx (3)</td>
<td>Intel®486™ SL processor</td>
</tr>
<tr>
<td>00</td>
<td>0010</td>
<td>0101</td>
<td>xxx (1)</td>
<td>Intel®SX2™ processors</td>
</tr>
<tr>
<td>00</td>
<td>0010</td>
<td>0111</td>
<td>xxx (3)</td>
<td>Write-Back Enhanced Intel®DX2™ processors</td>
</tr>
<tr>
<td>00</td>
<td>0010</td>
<td>1000</td>
<td>xxx (3)</td>
<td>Intel®DX4™ processors</td>
</tr>
<tr>
<td>00, 01</td>
<td>0100</td>
<td>1000</td>
<td>xxx (3)</td>
<td>Intel®DX4™ OverDrive® processors</td>
</tr>
<tr>
<td>00</td>
<td>0101</td>
<td>0001</td>
<td>xxx (2)</td>
<td>Pentium® processors (60, 66)</td>
</tr>
<tr>
<td>00</td>
<td>0101</td>
<td>0010</td>
<td>xxx (2)</td>
<td>Pentium® processor for Pentium processor (60, 66)</td>
</tr>
<tr>
<td>01</td>
<td>0101</td>
<td>0010</td>
<td>xxx (2)</td>
<td>Pentium® processor for Pentium processor (75, 90, 100, 120, 133)</td>
</tr>
<tr>
<td>01</td>
<td>0101</td>
<td>0011</td>
<td>xxx (2)</td>
<td>Pentium® OverDrive® processors for Intel486™ processor-based systems</td>
</tr>
<tr>
<td>00</td>
<td>0101</td>
<td>0100</td>
<td>xxx (2)</td>
<td>Pentium® processor with MMX™ technology (166, 200)</td>
</tr>
<tr>
<td>01</td>
<td>0101</td>
<td>0100</td>
<td>xxx (2)</td>
<td>Pentium® OverDrive® processor with MMX™ technology for Pentium processor</td>
</tr>
<tr>
<td>00</td>
<td>0110</td>
<td>0001</td>
<td>xxx (2)</td>
<td>Pentium® Pro processor</td>
</tr>
<tr>
<td>00</td>
<td>0110</td>
<td>0011</td>
<td>xxx (2)</td>
<td>Pentium® II processor, model 3</td>
</tr>
<tr>
<td>00</td>
<td>0110</td>
<td>0101(5)</td>
<td>xxx (2)</td>
<td>Pentium® II processor, model 5, Pentium II Xeon™ processor and Intel® Celeron™ processor, model 5</td>
</tr>
<tr>
<td>00</td>
<td>0110</td>
<td>0110</td>
<td>xxx (2)</td>
<td>Intel® Celeron™ processor, model 6</td>
</tr>
<tr>
<td>00</td>
<td>0110</td>
<td>0111(6)</td>
<td>xxx (2)</td>
<td>Intel® Pentium® III processor and Intel Pentium III Xeon™ processor</td>
</tr>
<tr>
<td>01</td>
<td>0110</td>
<td>0011</td>
<td>xxx (2)</td>
<td>Intel®Pentium® II OverDrive® processor</td>
</tr>
</tbody>
</table>
NOTES:
1. This processor does not implement the CPUID instruction.
2. Refer to the Intel486™ documentation, the Pentium® Processor Specification Update (Order Number 242480), the Pentium® Pro Processor Specification Update (Order Number 242889), the Pentium® II Processor Specification Update (Order Number 243337), the Pentium® II Xeon™ Processor Specification Update (Order Number 243776), the Intel® Celeron™ Processor Specification Update (Order Number 243748), the Pentium® III Processor Specification Update (Order Number 244453) or the Pentium® III Xeon™ Processor Specification Update (Order Number 244460) for the latest list of stepping numbers.
3. Stepping 3 implements the CPUID instruction.
4. The definition of the type field for the OverDrive® processor is 01h. An errata on the Pentium® OverDrive processor will always return 00h as the type.
5. To differentiate between the Pentium® II processor, model 5, Pentium II Xeon™ processor and the Intel® Celeron™ processor, model 5, software should check the cache descriptor values through executing CPUID instruction with EAX = 2. If no L2 cache is returned, the processor is identified as an Intel Celeron processor, model 5. If 1M or 2M L2 cache size is reported, the processor is the Pentium II Xeon processor otherwise it is a Pentium II processor, model 5 or a Pentium II Xeon processor with 512K L2 cache size.
6. To differentiate between the Pentium® III processor and the Pentium® III Xeon™ processor, software should check the cache descriptor values through executing CPUID instruction with EAX = 2. If 1M or 2M L2 cache size is reported, the processor is the Pentium III Xeon processor otherwise it is a Pentium III processor or a Pentium III Xeon processor with 512K L2 cache size.
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. 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.

![Figure 4. Processor Signature Format on Intel® 386™ Processors](image-url)
Table 4. Intel386™ Processor Signatures

<table>
<thead>
<tr>
<th>Type</th>
<th>Family</th>
<th>Major Stepping</th>
<th>Minor Stepping</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0011</td>
<td>0000</td>
<td>xxxx</td>
<td>Intel386™ DX processor</td>
</tr>
<tr>
<td>0010</td>
<td>0011</td>
<td>0000</td>
<td>xxxx</td>
<td>Intel386™ SX processor</td>
</tr>
<tr>
<td>0010</td>
<td>0011</td>
<td>0000</td>
<td>xxxx</td>
<td>Intel386™ CX processor</td>
</tr>
<tr>
<td>0010</td>
<td>0011</td>
<td>0000</td>
<td>xxxx</td>
<td>Intel386™ EX processor</td>
</tr>
<tr>
<td>0100</td>
<td>0011</td>
<td>0000 and 0001</td>
<td>xxxx</td>
<td>Intel386™ SL processor</td>
</tr>
<tr>
<td>0000</td>
<td>0011</td>
<td>0100</td>
<td>xxxx</td>
<td>RapidCAD® coprocessor</td>
</tr>
</tbody>
</table>

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</td>
<td>Reserved</td>
<td>Do not count on their value.</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>SEP</td>
<td>Fast System Call</td>
<td>Indicates whether the processor supports the Fast System Call instructions, SYSENTER and SYSEXIT. NOTE: Refer to Section 3.5 for further information regarding SYSENTER/ SYSEXIT feature and SEP feature bit.</td>
</tr>
<tr>
<td>12</td>
<td>MTRR</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 MC_G_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 FCOMVCC and FCOMI instructions.</td>
</tr>
<tr>
<td>16</td>
<td>PAT</td>
<td>Page Attribute Table</td>
<td>Indicates whether the processor supports the Page Attribute Table. This feature augments the Memory Type Range Registers (MTRRs), allowing an operating system to specify attributes of memory on a 4K granularity through a linear granularity.</td>
</tr>
</tbody>
</table>
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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>PSE-36</td>
<td>36-bit Page Size Extension</td>
<td>Indicates whether the processor supports 4-Mbyte pages that are capable of addressing physical memory beyond 4GB. This feature indicates that the upper four bits of the physical address of the 4-Mbyte page is encoded by bits 13-16 of the page directory entry.</td>
</tr>
<tr>
<td>18</td>
<td>Processor serial number</td>
<td>Processor serial number is present and enabled</td>
<td>The processor supports the 96-bit processor serial number feature, and the feature is enabled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19 – 22</td>
<td>Reserved</td>
<td>Do not count on their value.</td>
<td></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</td>
<td>FXSR</td>
<td>Fast floating point save and restore</td>
<td>Indicates whether the processor supports the FXSAVE and FXRSTOR instructions for fast save and restore of the floating point context. Presence of this bit also indicates that CR4.OSFXSR is available for an operating system to indicate that it uses the fast save/restore instructions.</td>
</tr>
<tr>
<td>25</td>
<td>Streaming SIMD Extension</td>
<td>Streaming SIMD Extensions supported</td>
<td>The processor supports the Streaming SIMD Extensions to the Intel Architecture.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26 – 31</td>
<td>Reserved</td>
<td>Do not count on their value.</td>
<td></td>
</tr>
</tbody>
</table>

NOTE:
1. The processor contains a software-accessible Local APIC.

The remainder of the EAX register, and the EBX, ECX, and EDX registers, contain valid 8-bit descriptors. Table 6 shows that when bit 31 is zero, the register contains valid 8-bit descriptors. 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

<table>
<thead>
<tr>
<th>Register bit 31</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>

3.5. SYSENTER/SYSEXIT – SEP Features Bit

The presence of this facility is indicated by the SYSENTER Present (SEP) bit 11 of CPUID. An operating system that detects the presence of the SEP bit must also qualify the processor family and model to ensure that the SYSENTER/SYSEXIT instructions are actually present:

```c
If (CPUID SEP bit is set) {
    If (Family == 6) AND (Model < 3) AND (Stepping < 3) {  
        Fast System Call is NOT supported
    } ELSE  
        Fast System Call is supported
}
```

The Pentium® Pro processor (Model = 1) returns a set SEP CPUID feature bit, but should not be used by software.
3.6. Pentium® Pro Processor Output Example

The Pentium® Pro processor 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 fully associative, and has 2 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>
<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, 4K pages, 4-way set associative, 32 entries</td>
</tr>
<tr>
<td>02h</td>
<td>Instruction TLB, 4M pages, fully associative, 2 entries</td>
</tr>
<tr>
<td>03h</td>
<td>Data TLB, 4K pages, 4-way set associative, 64 entries</td>
</tr>
<tr>
<td>04h</td>
<td>Data TLB, 4M pages, 4-way set associative, 8 entries</td>
</tr>
<tr>
<td>06h</td>
<td>Instruction cache, 8K, 4-way set associative, 32 byte line size</td>
</tr>
<tr>
<td>08h</td>
<td>16KB instruction cache, 4-way set associative, 32 byte line size</td>
</tr>
<tr>
<td>0Ah</td>
<td>Data cache, 8K, 2-way set associative, 32 byte line size</td>
</tr>
<tr>
<td>0Ch</td>
<td>16KB data cache, 4-way set associative, 32 byte line size</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, 128K</td>
</tr>
<tr>
<td>42h</td>
<td>Unified cache, 32 byte cache line, 4-way set associative, 256K</td>
</tr>
<tr>
<td>43h</td>
<td>Unified cache, 32 byte cache line, 4-way set associative, 512K</td>
</tr>
<tr>
<td>44h</td>
<td>Unified cache, 32 byte cache line, 4-way set associative, 1M</td>
</tr>
<tr>
<td>45h</td>
<td>Unified cache, 32 byte cache line, 4-way set associative, 2M</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>EAX</th>
<th>03h</th>
<th>02h</th>
<th>01h</th>
<th>01h</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ECX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EDX</td>
<td>06h</td>
<td>04h</td>
<td>0Ah</td>
<td>42h</td>
</tr>
</tbody>
</table>

INTEL CONFIDENTIAL
(until publication date)
3.7. Pentium® II Processor, model 3 Output Example

The Pentium® II processor, model 3 returns the values shown in Table 9. If the value of AL=1, it is valid to interpret the remainder of the registers according to Table 7. Table 9 also shows the MSB of EAX register is 0. As with the Pentium Pro processor this indicates the upper 8 bits constitute an 8 bit descriptor. The remaining register values in Table 9 shows the Pentium II 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 fully associative, and has 2 entries.
- An instruction TLB that maps 4K pages, is 4 way set associative, and has 32 entries.
- A data cache that is 16K, is 4 way set associative, and has 32 entries.
- A data TLB that maps 4M pages, is 4 way set associative, and has 8 entries.
- An instruction cache that is 16K, is 4 way set associative, and has a 32 byte line size.
- A unified cache that is 512K, is 4 way set associative, and has a 32 byte line size.

4.0. PROCESSOR SERIAL NUMBER

The Pentium® III processors, and the Pentium III Xeon™ processors extend the concept of processor identification with the addition of processor serial number. Processor serial number is a 96-bit number accessible through the CPUID instruction. Processor serial number can be used by applications to identify a processor, and by extension, its system.

The processor serial number creates a software accessible identity for an individual processor. The processor serial number, combined with other qualifiers, could be applied to user identification. Applications include, membership authentication, data backup/restore protection, removable storage data protection, managed access to files, or to confirm document exchange between appropriate users.

Processor serial number is another tool for use in asset management, product tracking, remote systems load and configuration, or to aid in boot-up configuration. In the case of system service, processor serial number could be used to differentiate users during help desk access, or track error reporting.

Processor serial number provides an identifier for the processor, but should not be assumed to be unique in itself. There are potential modes in which erroneous processor serial numbers may be reported. For example, in the event a processor is operated outside its recommended operating specifications, the processor serial number may not be correctly read from the processor. Improper BIOS or software operations could yield an inaccurate processor serial number. These events could lead to possible erroneous or duplicate processor serial numbers being reported. System manufacturers can strengthen the robustness of the feature by including redundancy features, or other fault tolerant methods.

Processor serial number used as a qualifier for another independent number could be used to create an electrically accessible number which is likely to be distinct. Processor serial number is one building block useful for the purpose of enabling the trusted, connected PC.

### Table 9. Pentium® II Processor, model 3 with 512K L2 Cache, CPUID (EAX=2) Example Return Values

<table>
<thead>
<tr>
<th>EAX</th>
<th>03h</th>
<th>02h</th>
<th>01h</th>
<th>01h</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ECX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EDX</td>
<td>0Ch</td>
<td>04h</td>
<td>08h</td>
<td>43h</td>
<td></td>
</tr>
</tbody>
</table>
4.1. Presence of Processor Serial Number

To determine if the processor serial number feature is supported, the program should set the EAX register parameter value to “1” and then execute the CPUID instruction as follows:

```assembly
MOV EAX, 01H
CPUID
```

After execution of the CPUID instruction, the EDX register contains the Feature Flags. If Feature Flags bit 18 equals “1”, then the processor serial number feature is supported, and enabled. If Feature Flags bit 18 equals “0”, the processor serial number feature is either not supported, or disabled.

4.2. Forming the 96-bit Processor Serial Number

The 96-bit processor serial number is the concatenation of three 32-bit entities.

To access the most significant 32-bits of the processor serial number the program should set the EAX register parameter value to “1” and then execute the CPUID instruction as follows:

```assembly
MOV EAX, 01H
CPUID
```

After execution of the CPUID instruction, the EAX register contains the Processor Signature. The Processor Signature comprises the most significant 32-bits of the processor serial number. The value in EAX should be saved prior to gathering the remaining 64-bits of the processor serial number.

To access the remaining 64-bits of the processor serial number the program should set the EAX register parameter value to “3” and then execute the CPUID instruction as follows:

```assembly
MOV EAX, 03H
CPUID
```

After execution of the CPUID instruction, the EDX register contains the middle 32-bits, and the ECX register contains the least significant 32-bits of the processor serial number. Software may then concatenate the saved Processor Signature, EDX, and ECX before returning the complete 96-bit processor serial number.

Processor serial number should be displayed as 6 groups of 4 hex nibbles (Ex. XXXX-XXXX-XXXX-XXXX-XXXX-XXXX where X represents a hex digit). Alpha hex characters should be displayed as capital letters.

5.0. 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 6.0.
- 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 (P6 family 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 steppings or features. For example, the Intel® 386™ processor A-step had bit instructions that were withdrawn with the 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 Intel® 486™ processor used the same opcodes for different instructions. The software should have used the stepping information in the processor signature.
- Test feature flags individually and do not make assumptions about undefined bits. For example,
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 processor speed-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 a combination of the system’s timers to measure elapsed time and the TSC (Time Stamp Counter) to measure processor core clocks to allow direct calibration of the processor core.

- 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 not rely on the result of the CPUID algorithm when executed in virtual 8086 mode.

- Do not assume any ordering of model and/or stepping numbers. They are assigned arbitrarily.

- Do not assume processor serial number is a unique number without further qualifiers.

- Display processor serial number as 6 groups of 4 hex nibbles (Ex. XXXX-XXXX-XXXX-XXXX-XXXX-XXXX where X represents a hex digit).

- Display alpha hex characters as capital letters.

6.0. PROPER IDENTIFICATION SEQUENCE

The cpuid3a.asm program example demonstrates the correct use of the CPUID instruction. (See Example 1.) It also shows how to identify earlier processor generations that do not implement the processor signature or CPUID instruction. (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, Pentium II processors, Pentium II Xeon™ processors, Pentium II OverDrive® processors, Intel® Celeron™ processors, Pentium III processors, and Pentium III Xeon 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.
Is it an 8086 processor?  
Yes: cpu_type=0  
No:  
Is it an 80286 processor?  
Yes: cpu_type=2  
No:  
Is it an 80386 processor?  
Yes: cpu_type=3  
No: cpu_type=4  
Is the CPUID instruction supported?  
Yes: cpuid_flag = 1; indicates CPUID instruction present. Execute CPUID with input of 0 to get vendor ID string and input values for EAX.  
No:  
Does the vendor ID = GenuineIntel?  
Yes: 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.  
No:  
end_get_cpu_type

Figure 5. Flow of Processor get_cpu_type Procedure
7.0. USAGE PROGRAM EXAMPLES

The cpuid3b.asm or cpuid3.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 cpuid3.c example is written in the C language (see Example 2 and Example 3). Figure 6 presents an overview of the relationship between the three program examples.

![Flowchart](000964)

Figure 6. Flow of Processor Identification Extraction Procedure
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™ family processor
4=Intel® 486™ family processor
5=Pentium® family processor
6=P6 family of processors

_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 Microsoft* Developer Studio. This code correctly detects the current Intel ®8086/8088, 80286, 80386, 80486, Pentium®processor, PentiumPro processor, Pentium II processor, Pentium II Xeon™ processor, Pentium II Overdrive®, Intel® Celeron processor, Pentium III processor and Pentium III Xeon processor in the real-address mode only.

NOTE: When using this code with C program cpuid3.c, 32-bit segments are recommended.

To assemble this code with TASM, add the JUMPS directive.
jumps ; Uncomment this line for TASM

TITLE cpuid3a

comment this line for 32-bit segments

DOSSEG

1/14/99 2:13 PM CPUID

INTEL CONFIDENTIAL
(until publication date)
uncomment the following 2 lines for 32-bit segments

; .386
; .model flat

comment this line for 32-bit segments

; .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
public _cache_eax
public _cache_ebx
public _cache_ecx
public _cache_edx
public _sep_flag
public _platform_id

_cpu_type db 0
_fpu_type db 0
_v86_flag db 0
_cpuid_flag db 0
_intel_CPU db 0
_vendor_id db "------------"
_vendor_id db "GenuineIntel"
_intel_id db "GenuineIntel"

.code

; comment this line for 32-bit segments

; .8086
22
uncomment this line for 32-bit segments

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.

For 32-bit segments comment the following lines down to the next
comment line that says “STOP”

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 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.

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.

.check_386:

"STOP"

; 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 ; processor=80386
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.

.check_486:

check_80486:

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

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

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

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

cmp dword ptr intel_id, ebx
jne end_cpuid_type ; if not equal, not an Intel processor
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 ; 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

; Execute CPUID instruction to determine the cache descriptor
; information.

mov eax, 0 ; set up to check the EAX value
CPU_ID ; set up to check the EAX value
cmp ax, 2 ; Are cache descriptors supported?
jl end_cpuid_type

mov eax, 2 ; set up to read cache descriptor
CPU_ID ; set up to read cache descriptor
cmp al, 1 ; Is one iteration enough to obtain
jne end_cpuid_type ; cache information?
; This code supports one iteration
; only.

mov _cache_eax, eax ; store cache information
mov _cache_ebx, ebx ; NOTE: for future processors, CPUID
mov _cache_ecx, ecx ; instruction may need to be run more
mov _cache_edx, edx ; than once to get complete cache

end_cpuid_type:
pop edi ; restore registers
pop esi
pop ebx

; comment this line for 32-bit segments

1/14/99 2:13 PM  CPUID

INTEL CONFIDENTIAL
(unti...
.8086
end_cpu_type:
    ret
_get_cpu_type   endp

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

public _get_fpu_type
_get_fpu_type proc

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

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

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

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

; 80287/80387 check for the Intel386 processor

check_infinity:
    cmp _cpu_type, 3 ; must use default control from FNINIT
    jne end_fpu_type
    fld1 ; form infinity
    fldz ; 8087/Intel287 NDP say +inf = -inf
    fdiv ; 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
    sahf ; store Intel287 NDP for FPU type
    ; see if infinities matched

end_fpu_type:
Example 2. Processor Identification Procedure in Assembly Language

Filename: cpuid3b.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 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 Microsoft Developer Studio. 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®, Pentium® Pro, Pentium® II processors, Pentium® II Xeon processors, Pentium II Overdrive processors, Intel Celeron™ processors, Pentium III processors and Pentium III Xeon processors in the real-address mode.

NOTE: This code is written using 16-bit Segments
To assemble this code with TASM, add the JUMPS directive.
TITLE cpuid3b
DOSSEG
.model small
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.

```
.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 _print

mov ax, 4c00h
int 21h
```

```
extrn __get_cpu_type: proc
```

```
extrn __get_fpu_type: proc
```

```
FPU_FLAG equ 0001h
```

AP-485

INTEL CONFIDENTIAL

(unti, publication date)
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
SEP_FLAG equ 0800h
MTRR_FLAG equ 1000h
PGE_FLAG equ 2000h
MCA_FLAG equ 4000h
CMOV_FLAG equ 8000h
PAE_FLAG equ 10000h
MSR_FLAG equ 20000h
PSE36_FLAG equ 40000h
MCA_FLAG equ 80000h
SIMD_FLAG equ 100000h
PPU_FLAG equ 200000h

.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$"
cp_486 db "n 80486DX, 80486DX2 processor or“
cp_486sx db "n 80486sx processor$"
fp_8087 db "n 8087 math coprocessor$"
fp_287 db "n 80287 math coprocessor$"
fp_387 db "n 80387 math coprocessor$"
int486_msg db "Genuine Intel486(TM) processor$"
int486dx_msg db "Genuine Intel486(TM) DX processor$"
int486sx_msg db "Genuine Intel486(TM) SX processor$"
inteldx2_msg db "Genuine IntelDX2(TM) processor$"
inteldx2wb_msg db "Genuine Write-Back Enhanced“
inteldx2wb_msg db "IntelDX2(TM) processor$"
pentium_msg db "Genuine Intel Pentium(TM) processor$"
pentiumiimodel3_msg db "Genuine Intel Pentium(TM) II processor, model 3$"
pentiumiimodel3_msg db "Genuine Intel Pentium(TM) II processor, model 5 or Intel Pentium(TM) II Xeon(TM) processor$"
pentiumixeon_m5_msg db "Genuine Intel Pentium(TM) II processor, model 5 or Intel Pentium(TM) II Xeon(TM) processor$"
ppu_flag db "Genuine Intel Pentium(TM) II processor or Intel Pentium(TM) II Xeon(TM) processor$"
ppu_flag db "Genuine Intel Pentium(TM) III processor or Intel Pentium(TM) II Xeon(TM) processor$"
ppu_flag db "Genuine Intel Pentium(TM) III processor or Intel Pentium(TM) II Xeon(TM) processor$"
unknown_msg db "n unknown Genuine Intel processor$"
; The following 16 entries must stay intact as an array
intel_486_0  dw offset intel486dx_msg
intel_486_1  dw offset intel486dx_msg
intel_486_2  dw offset intel486sx_msg
intel_486_3  dw offset inteldx2_msg
intel_486_4  dw offset intel486_msg
intel_486_5  dw offset intelsx2_msg
intel_486_6  dw offset intel486_msg
intel_486_7  dw offset inteldx2wb_msg
intel_486_8  dw offset inteldx4_msg
intel_486_9  dw offset intel486_msg
intel_486_a  dw offset intel486_msg
intel_486_b  dw offset intel486_msg
intel_486_c  dw offset intel486_msg
intel_486_d  dw offset intel486_msg
intel_486_e  dw offset intel486_msg
intel_486_f  dw offset intel486_msg
; end of array

family_msg  db 13,10,"Processor Family: $"
model_msg  db 13,10,"Model: $"
stepping_msg db 13,10,"Stepping: "
cr_LF  db 13,10,"$"
turbo_msg db 13,10,"The processor is an OverDrive(R) processor"
dp_msg db 13,10,"The processor is the upgrade processor in a dual processor system$"
fpu_msg db 13,10,"The processor contains an on-chip FPU$"
vme_msg db 13,10,"The processor supports Virtual Mode Extensions$"
de_msg db 13,10,"The processor supports Debugging Extensions$"
pse_msg db 13,10,"The processor supports Page Size Extensions$"
tsc_msg db 13,10,"The processor supports Time Stamp Counter$"
msr_msg db 13,10,"The processor supports Model Specific Registers$"
pae_msg db 13,10,"The processor supports Physical Address Extensions$"
mce_msg db 13,10,"The processor supports Machine Check Exceptions$"
cx8_msg db 13,10,"The processor supports the CMPXCHG8B instruction$"
apic_msg db 13,10,"The processor contains an on-chip APIC$"
sep_msg db 13,10,"The processor supports Fast System Call$"
no_sep_msg db 13,10,"The processor does not support Fast System Call$"
mtrr_msg db 13,10,"The processor supports Memory Type Range Registers$"
pge_msg db 13,10,"The processor supports Page Global 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$"
pat_msg  db 13,10,"The processor supports Page  Attribute"
         db “ Table$"
pse36_msg db 13,10,"The processor supports 36-bit Page"
         db “ Size Extension$"
psnum_msg db 13,10,"The processor supports the"
         db “ processor serial number$”
mmx_msg  db 13,10,"The processor supports Intel Architecture”
         db “ MMX(TM) Technology$”
fxsr_msg db 13,10,"The processor supports Fast floating point”
         db “ save and restore$”
simd_msg db 13,10,"The processor supports the Streaming:”
         db “ SIMD extensions to the Intel Architecture$”
not_intel db “t least an 80486 processor,”
         db 13,10,”It does not contain a Genuine”
         db “Intel part and as a result,”
         db “the”,13,10,” CPUID”
         db “ detection information cannot be”
         db “ determined at this time.$”

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 _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
    jmp 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 fp_486

print_486sx:
    mov ah, 9h
    int 21h
    jmp end_print
print_unknown:
    mov dx, offset cp_error
    jmp print_486sx

print_cpuid_data:
    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_486_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 & model 1, print Pentium Pro processor
    jne print_pentiumpro_type
    mov ax, word ptr _cpu_signature
    shr ax, 4
    and eax, 0fh ; isolate model
    cmp eax, 3
    jge print_pentiumiimodel3_type ; incorrect model number = 2
    cmp eax, 1
    jne print_unknown_type ; incorrect model number = 2
    mov _sep_flag, 0 ; does not support Fast System Call
    mov dx, offset pentiumpro_msg
    jmp print_common

print_pentiumiimodel3_type:
    cmp eax, 3 ; if 6 & model 3, print Pentium II processor, model 3
    jne print_pentiumiimodel5_type
    mov ax, word ptr _cpu_signature
    and al, 0fh ; isolate stepping
    cmp al, 3
    jl no_sep
    mov _sep_flag, 1
    mov dx, offset pentiumiimodel3_msg
    jmp print_common

no_sep:
    mov _sep_flag, 0 ; stepping does not support Fast System Call
    mov dx, offset pentiumiimodel3_msg
jmp print_common

print_pentiumimodel5_type:
cmp eax, 5
; if 6 & model 5, either Pentium II processor, model 5, Pentium II Xeon processor or Intel Celeron processor, model 5
je celeron_xeon_detect
cmp eax, 7
; If model 7 check cache descriptors to determine Pentium III or Pentium III Xeon
jne print_celeronmodel6_type

CELERON XEON DETECT:
mov _sep_flag, 1
; Pentium II processor, model 5, Pentium II Xeon processor, Pentium III processor or Pentium III Xeon processor, Intel Celeron processor support sep flag

OP_O
mov ax, word ptr _cache_eax
OP_O
rol ax, 8
mov cx, 3

CELERON DETECT EAX:
cmp al, 40h
; Is it no L2
je print_celeron_type
jal print_pentiumiixeon_type
; Is L2 >= 1M
OP_O
rol ax, 8
loop celeron_detect_eax

OP_O
mov ax, word ptr _cache_ebx
mov cx, 4

CELERON DETECT EBX:
cmp al, 40h
; Is it no L2
je print_celeron_type
jal print_pentiumiixeon_type
; Is L2 >= 1M
OP_O
rol ax, 8
loop celeron_detect_ebx

OP_O
mov ax, word ptr _cache_ecx
mov cx, 4
celeron_detect_ecx:
  cmp al, 40h ; Is it no L2
  je print_celeron_type
  cmp al, 44h ; Is L2 >= 1M
  jae print_pentiumiixeon_type

  OP_O
  rol ax, 8
  loop celeron_detect_ecx

  OP_O
  mov ax, word ptr _cache_edx
  mov cx, 4

celeron_detect_edx:
  cmp al, 40h ; Is it no L2
  je print_celeron_type
  cmp al, 44h ; Is L2 >= 1M
  jae print_pentiumiixeon_type

  OP_O
  rol ax, 8
  loop celeron_detect_edx

  mov dx, offset pentiumiixeon_m5_msg
  mov ax, word ptr _cpu_signature
  shr ax, 4
  and eax, 0fh ; isolate model
  cmp eax, 5 ; if 6 & model 6, print Intel Celeron model 6
  jne print_unknown_type
  mov ax, word ptr _cpu_signature
  and al, 0fh
  mov _sep_flag, 1 ; Intel Celeron processor, model 6 supports SEP
  jmp print_common

print_celeron_type:
  mov dx, offset celeron_msg
  jmp print_common

print_pentiumiixeon_type:
  mov dx, offset pentiumiixeon_msg
  mov ax, word ptr _cpu_signature
  shr ax, 4
  and eax, 0fh ; isolate model
  cmp eax, 5 ; if 6 & model 6, print Intel Celeron processor, model 6
  jne print_unknown_type
  mov dx, offset pentium_IIIxeon_msg
  jmp print_common

print_celeronmodel6_type:
  cmp eax, 6
  jne print_unknown_type
  mov ax, word ptr _cpu_signature
  and al, 0fh
  mov _sep_flag, 1 ; Intel Celeron processor, model 6 supports SEP
  mov dx, offset celeronmodel6_msg
  jmp print_common

1/14/99 2:13 PM  CPUID

INTEL CONFIDENTIAL
(unti1 publication date)
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:
  36
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_SEP
mov dx, offset apic_msg
mov ah, 9h
int 21h

check_SEP:
cmp _sep_flag, 1
jne print_no_sep
mov dx, offset sep_msg
mov ah, 9h
int 21h
jmp check_MTRR

print_no_sep:
mov dx, offset no_sep_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_PAT
mov dx, offset cmov_msg
mov ah, 9h
int 21h

check_PAT:
mov eax, dword ptr _features_edx
and eax, PAT_FLAG ; check for PAT
jz check_PSE36

38
mov dx, offset pat_msg
mov ah, 9h
int 21h

check_PSE36:
    mov eax, dword ptr _features_edx
    and eax, PSE36_FLAG
    jz check_PSNUM
    mov dx, offset pse36_msg
    mov ah, 9h
    int 21h

check_PSNUM:
    mov eax, dword ptr _features_edx
    and eax, PSNUM_FLAG
    ; check for processor serial number
    jz check_MMX
    mov dx, offset psnum_msg
    mov ah, 9h
    int 21h

check_MMX:
    mov eax, dword ptr _features_edx
    and eax, MMX_FLAG
    ; check for MMX technology
    jz check_FXSR
    mov dx, offset mmx_msg
    mov ah, 9h
    int 21h

check_FXSR:
    mov eax, dword ptr _features_edx
    and eax, FXSR_FLAG
    ; check for FXSR
    jz check_SIMD
    mov dx, offset fxsr_msg
    mov ah, 9h
    int 21h

check_SIMD:
    mov eax, dword ptr _features_edx
    and eax, SIMD_FLAG
    ; check for Streaming SIMD
    jz end_print
    ; Extensions
    mov dx, offset simd_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: cpuid3.c */
/* */
/* 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 Developer Studio. */
/* 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), Pentium(R) Pro, */
/* Pentium® II, Pentium(R) II Xeon, Pentium(R) II OverDrive(R), */
/* Intel Celeron, Intel Pentium III and Intel Pentium III Xeon 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 SEP_FLAG 0x0800
#define MTRR_FLAG 0x0080
#define PGE_FLAG 0x0000
#define MCA_FLAG 0x0040
#define CMOV_FLAG 0x0000
#define PAT_FLAG 0x0100
#define PSE36_FLAG 0x0200
#define PSNUM_FLAG 0x0000
#define MMX_FLAG 0x0000
#define FXSR_FLAG 0x0000
#define SIMD_FLAG 0x0000

1/14/99 2:13 PM  CPUID

INTEL CONFIDENTIAL
(until publication date)
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;
extern long cache_eax;
extern long cache_ebx;
extern long cache_ecx;
extern long cache_edx;

long cache_temp;
long celeron_flag;
long pentiumxeon_flag;

main() {
    get_cpu_type();
    get_fpu_type();
    print();
}

print() {
    printf("This system has a");
    if (cpuid_flag == 0) {
        switch (cpu_type) {
            case 0:
                printf(" an 8086/8088 processor");
                if (fpu_type) printf(" and an 8087 math coprocessor");
                break;
            case 2:
                printf(" an 80286 processor");
                if (fpu_type) printf(" and an 80287 math coprocessor");
                break;
            case 3:
                printf(" an 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(" 80486DX, 80486DX2 processor or 80487SX math coprocessor");
                else
                    printf(" 80486SX processor");
                break;
            default:
                printf(" unknown processor");
        }
    }
    else {
        /* using cpuid instruction */
    }
}
if (intel_CPU) {
    if (cpu_type == 4) {
        switch (((cpu_signature >> 4) & 0xf)) {
            case 0:
                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");
                break;
        }
    } else if (cpu_type == 5)
        printf("Genuine Intel Pentium(R) processor");
    else if (((cpu_type == 6) && (((cpu_signature >> 4) & 0xf) == 1))
        printf("Genuine Intel Pentium(R) Pro processor");
    else if (((cpu_type == 6) && (((cpu_signature >> 4) & 0xf) == 3))
        printf("Genuine Intel Pentium(R) II processor, model 3");
    else if (((cpu_type == 6) && (((cpu_signature >> 4) & 0xf) == 5))
        printf("Genuine Intel Pentium(R) II processor, model 5");
    else if (((cpu_type == 6) && (((cpu_signature >> 4) & 0xf) == 7))
        printf("Genuine Intel Pentium(R) II processor, model 7");
}

CELERON_FLAG = 0;
PENTIUMXEON_FLAG = 0;
cache_temp = cache_eax & 0xFF000000;
if (cache_temp == 0x40000000) celeron_flag = 1;
if (cache_temp >= 0x44000000) && (cache_temp <= 0x45000000) peniumx86eon_flag = 1;
cache_temp = cache_eax & 0xFF000000;
if (cache_temp == 0x40000000) celeron_flag = 1;
if (cache_temp >= 0x44000000) && (cache_temp <= 0x45000000) peniumx86eon_flag = 1;
cache_temp = cache_eax & 0xFF000000;
if (cache_temp == 0x40000000) celeron_flag = 1;
if (cache_temp >= 0x44000000) && (cache_temp <= 0x45000000) peniumx86eon_flag = 1;
cache_temp = cache_eax & 0xFF000000;
if (cache_temp == 0x40000000) celeron_flag = 1;
if (cache_temp >= 0x44000000) && (cache_temp <= 0x45000000) peniumx86eon_flag = 1;
pentiumxeon_flag = 1;

cache_temp = cache_ebx & 0xFF000000;
if (cache_temp == 0x40000000)
celeron_flag = 1;
if ((cache_temp >= 0x44000000) && (cache_temp <= 0x45000000))
pentiumxeon_flag = 1;

cache_temp = cache_ebx & 0xFF000000;
if (cache_temp == 0x40000000)
celeron_flag = 1;
if ((cache_temp >= 0x44000000) && (cache_temp <= 0x45000000))
pentiumxeon_flag = 1;

cache_temp = cache_ebx & 0xFF000000;
if (cache_temp == 0x40000000)
celeron_flag = 1;
if ((cache_temp >= 0x44000000) && (cache_temp <= 0x45000000))
pentiumxeon_flag = 1;
cache_temp = cache_edx & 0xFF0000;
if (cache_temp == 0x400000)
    celeron_flag = 1;
if ((cache_temp >= 0x400000) && (cache_temp <= 0x450000))
    pentiumxeon_flag = 1;
cache_temp = cache_edx & 0xFF00;
if (cache_temp == 0x4000)
    celeron_flag = 1;
if ((cache_temp >= 0x4400) && (cache_temp <= 0x4500))
    pentiumxeon_flag = 1;
cache_temp = cache_edx & 0xFF;
if (cache_temp == 0x40)
    celeron_flag = 1;
if ((cache_temp >= 0x44) && (cache_temp <= 0x45))
    pentiumxeon_flag = 1;

if (celeron_flag == 1)
    printf(" Genuine Intel Celeron(TM) processor, model 5");
else
    {
        if (pentiumxeon_flag == 1) {
            if (((cpu_signature >> 4) & 0x0f) == 5)
                printf(" Genuine Intel Pentium(R) II Xeon(TM) processor");
            else
                printf(" Genuine Intel Pentium(R) III Xeon(TM) processor");
        }
        else {
            if (((cpu_signature >> 4) & 0x0f) == 5) {
                printf(" Genuine Intel Pentium(R) II processor, model 5");
                printf(" or Intel Pentium(R) II Xeon processor");
            }
            else {
                printf(" Genuine Intel Pentium(R) III processor,");
                printf(" or Intel Pentium(R) III Xeon(TM) processor");
            }
        }
    }
else if ((cpu_type == 6) && (((cpu_signature >> 4) & 0x0f) == 6))
    printf(" Genuine Intel Celeron(TM) processor, model 6");
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) 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("The processor supports the Debugging Extensions");
if (features_edx & PSE_FLAG)
  printf("The processor supports Page Size Extensions");
if (features_edx & TSC_FLAG)
  printf("The processor supports Time Stamp Counter");
if (features_edx & MSR_FLAG)
  printf("The processor supports Model Specific Registers");
if (features_edx & PAE_FLAG)
  printf("The processor supports Physical Address Extension");
if (features_edx & MCE_FLAG)
  printf("The processor supports Machine Check Exceptions");
if (features_edx & CX8_FLAG)
  printf("The processor supports the CMPXCHG8B instruction");
if (features_edx & APIC_FLAG)
  printf("The processor contains an on-chip APIC");
if (features_edx & SEP_FLAG) {
  if ((cpu_type == 6) && (((cpu_signature >> 4) & 0xf) < 3)
      && ((cpu_signature & 0xf) < 3))
    printf("The processor does not support the Fast System Call");
  else
    printf("The processor supports the Fast System Call");
}
if (features_edx & MTRR_FLAG)
  printf("The processor supports the Memory Type 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 Architecture");
if (features_edx & APIC_FLAG)
  printf("The processor supports the Conditional Move Instruction");
if (features_edx & PAT_FLAG)
  printf("The processor supports the Page Attribute Table");
if (features_edx & PSE36_FLAG)
  printf("The processor supports 36-bit Page Size Extension");
if (features_edx & PSNUM_FLAG)
  printf("The processor supports the processor serial number");
if (features_edx & FXSR_FLAG)
  printf("The processor supports the Fast floating point save and restore");
if (features_edx & SIMD_FLAG)
  printf("The processor supports the Streaming SIMD extensions to the Intel Architecture");
} else {
  printf("At least an 80486 processor.");
  printf("It does not contain a Genuine Intel part and as a result, the");
  printf("CPUID detection information cannot be determined at this time.");
}

printf("\n");