AMD

AMD Processor Recognition

Application Note

Publication # **20734** Rev: **Q**
Issue Date: **June 2000**

© 2000 Advanced Micro Devices, Inc. All rights reserved.

The contents of this document are provided in connection with Advanced Micro Devices, Inc. ("AMD") products. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product descriptions at any time without notice. No license, whether express, implied, arising by estoppel or otherwise, to any intellectual property rights is granted by this publication. Except as set forth in AMD's Standard Terms and Conditions of Sale, AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right.

AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.

Trademarks

AMD, the AMD logo, AMD Athlon, and combinations thereof, 3DNow!, K86, and AMD-K5 are trademarks, and AMD-K6, Am486, and $Am5_X86$ are registered trademarks of Advanced Micro Devices, Inc.

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation.

MMX is a trademark of Intel Corporation.

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Contents

List of Figures

List of Tables

Revision History

Date	Rev	Description
June 2000	Q	Added information about the AMD Athlon™ processor **Model 4** throughout the document. Revised "The AMD Athlon™ Processor Model 1, Model 2, and Model 4" on page 2. Revised Table 4, "Processor Name Strings for the AMD Athlon™ Processor," on page 13. Added code sample to "Code Samples" on page 14. Added "Displaying the AMD Athlon™ Processor Name String" on page 22 and "DisplayK7NameString Subroutine" on page 23. Revised information about bit 15 in Table 3, "Summary of Standard and Extended Feature Bits," on page 9. Revised name string for AMD Athlon™ processor Model 1 in Table 4, "Processor Name Strings for the AMD Athlon™ Processor," on page 13. Made Table 5, "Standard Feature Flag Descriptions for the AMD-K6®-2 and AMD-K6®-III Processors," on page 30 is specific to these processors. Added Table 6, "Standard Feature Flag Descriptions for the AMD Athlon™ Processors," on page 31, which is specific to these processors. Clarified instruction family and generation being derived from function 1 and function 8000_0001 respectively. Made Table 7, "Extended Feature Flag Descriptions for the AMD-K6®-2 and AMD-K6®-III Processors," on page 33 is specific to these processors. Added Table 8, "Extended Feature Flag Descriptions for the AMD Athlon™ Processors," on page 34, which is specific to these processors. Revised Table 21, "Values Returned By the AMD Athlon™ Processor," on page 42.
Dec 1999	P	Added the AMD Athlon processor **Model 2** information throughout document. Model 1 refers to the AMD Athlon processor manufactured with 0.25-micron process technology and Model 2 refers to the AMD Athlon processor manufactured with 0.18-micron process technology.
Nov 1999	O	Clarified usage of "Code Samples" on page 14. Added "Example Function Call" on page 22.
August 1999	N	Merged standard and extended feature bits into one table. See Table 3, "Summary of Standard and Extended Feature Bits," on page 9. Revised Table 21, "Values Returned By the AMD Athlon™ Processor," on page 42.

Date	Rev	Description
August 1999	M	Added the AMD Athlon™ processor information throughout document. Added url www.amd.com/products/cpg/bin, where codes samples and utilities are available. Revised "Testing for the CPUID Instruction" on page 4. Revised "Determining Instruction Set Support" on page 10. Revised Tables 9 through 19 to cross-reference new section—"Associativity Field Definitions" on page 40.
May 1999	L	In Table 11 on page 18, changed function 8000_0001h EDX entries for Models 6 and 7 from 0080_01BFh to 0080_05BFh.
May 1999	L	Added note about the name string for the AMD-K6-2 processor to Table 11 on page 18.
Feb 1999	K	Added L2 cache information to Table 1 on page 4. Added Function 8000_0006h to "Displaying Cache Information" on page 10. Added Function 8000_0006h — L2 Cache Information and Table 10, "ECX Format Returned by Function 8000_0006h," on page 17. Added AMD-K6-III processor Model 9 values and three notes to Table 11 on page 18.
Nov 1998	J	In "Standard Functions" on page 12, clarified AMD's vendor identification string stored in registers EBX, EDX, and ECX. In Table 11, "Values Returned By AMD-K6® Processors," on page 18, changed function 8000_0001h, EDX value for the AMD-K6 processor Model 7 and deleted note 2.
May 1998	I	Revised "Functions 8000_0002h, 8000_0003h, and 8000_0004h — Processor Name String" on page 16. Added return values for AMD-K6 processor Model 9 to Table 10 on page 18. Divided Appendix B table into two separate tables.
Jan 1998	H	Added revised bit 31 description and alternate test for AMD-K6-2 to "Identifying Supported Features" on page 6.
Dec 1997	G	Changed part names for AMD-K6 processor Models 8 and 9 in Table 2 on page 5. Added 3DNow!™ instructions feature (bit 31) to Table 4 on page 8 and Table 6 on page 15. Added AMD-K6®-2 processor return values to Table 12 on page 21.

Application Note

AMD Processor Recognition

Introduction

Due to the increasing number of choices available in the x86 processor marketplace, the need for a simple way for hardware and software to identify the type of processor and its feature set has become critical. The CPUID instruction was added to the x86 instruction set for this purpose. This document contains information about using the CPUID instruction to test for extended features, such as an L2 cache on the AMD-K6®-III processor, and AMD Athlon™ processor.

The CPUID instruction provides complete information about the processor (vendor, type, name, etc.) and its capabilities (features). After detecting the processor and its capabilities, software can be accurately tuned to the system for maximum performance and benefit to users. For example, software can roughly determine the performance level of a particular processor by detecting the type or speed of the processor. If the performance level is high enough, the software can enable additional capabilities or more advanced algorithms. Another example involves testing for the presence of 3DNow!™ or MMX™ instructions on the processor. If the software finds these features present when it checks the feature bits, it can utilize these more powerful extensions for dramatically better performance on new multimedia software. The code for determining the speed of AMD processors is located at the following url:

http://www.amd.com/products/cpg/bin

The AMD Athlon™ Processor Model 1, Model 2, and Model 4

This section contains information about Model 1, Model 2, and Model 4 of the AMD Athlon processor. Model 1 refers to the AMD Athlon processor manufactured with 0.25-micron process technology. Model 2 refers to the AMD Athlon processor manufactured with 0.18-micron process technology. Model 4 refers to the AMD Athlon processor containing a performance-enhancing internal 256-Kbyte L2 cache and manufactured with 0.18-micron process technology.

For more information, see Appendix A, "CPUID Instruction Definition" on page 27.

CPUID Instruction Overview

Software operating at any privilege level can execute the CPUID instruction to identify the processor and its feature set. In addition, the CPUID instruction implements multiple functions, each providing different information about the processor, including the vendor, model number, revision (stepping), features, cache organization, and processor name. The multiple-function approach allows the CPUID instruction to return a complete picture about the type of processor and its capabilities—more detailed information than could be returned by a single function. In addition to gathering all the information by calling multiple functions, the CPUID instruction provides the flexibility of making only one call to obtain the specific data requested.

The functions are divided into two types: **standard functions** and **extended functions**. Standard functions provide a simple method for software to access information common to all x86 processors. Extended functions provide information on extensions specific to a vendor's processor (for example, AMD family processors).

The flexibility of the CPUID instruction allows for the addition of new CPUID functions in future generations of processors. Appendix A, "CPUID Instruction Definition" on page 27 contains a detailed description of the CPUID instruction.

Testing for the CPUID Instruction

Beginning with the Am486®DX4 processor, all AMD family processors support the CPUID instruction. To use the CPUID instruction, software must first determine if the processor supports the CPUID instruction. CPUID support is determined in one of the following ways:

- Execute the CPUID instruction and check whether an illegal instruction exception occurs. If an exception occurs, the processor does not have CPUID support.
- Check if the ID bit (bit 21) of the EFLAGS register is writable. If the bit is writable (that is, it can be modified), the CPUID instruction is supported.

The operating system (OS) environment determines which approach is more appropriate. These techniques are described in the following sections.

Illegal Instruction Exception Method

This technique requires a way for a user program to detect and handle illegal instruction exceptions. Where such capabilities are present, this method represents a reliable way of detecting support for the CPUID instruction. The CPUID sample code starting on page 14 uses this approach.

EFLAGS ID-Bit Method

This technique retrieves the contents of EFLAGS using the PUSHFD instruction, toggles the ID bit, and uses the POPFD instruction to write the modified value of the ID bit into the EFLAGS register. It then retrieves the contents of EFLAGS using a second PUSHFD instruction and checks whether the value of the ID bit differs from the original value. If the value has changed, the CPUID instruction is available for identifying the processor and its features. The following code sample demonstrates the way a program uses the PUSHFD and POPFD instructions to test the ID bit.

```
pushfd                  ; Save EFLAGS to stack
pop   eax               ; Store EFLAGS in EAX
mov   ebx, eax          ; Save in EBX for testing later
xor   eax, 00200000h    ; Switch bit 21
push  eax               ; Copy changed value to stack
popfd                   ; Save changed EAX to EFLAGS
pushfd                  ; Push EFLAGS to top of stack
pop   eax               ; Store EFLAGS in EAX
cmp   eax, ebx          ; See if bit 21 has changed
jz    NO_CPUID          ; If no change, no CPUID
```

A potential problem with this approach is that an interrupt or a trap (such as a debug trap) can occur between the POPFD and the following PUSHFD, and that the interrupt or trap handler code destroys the value of the ID bit. Where possible, the above code should be preceded by a CLI instruction and followed by an STI instruction, which ensures that no interrupts occur between the POPFD and the PUSHFD. However, traps can still occur, even if the code is preceded by a CLI instruction and followed by an STI instruction.

Using CPUID Functions

When software uses the CPUID instruction to identify a processor, it is important that it uses the instruction appropriately. The instruction has been defined to make it easy to identify the type and features of x86 processors manufactured by many different vendors.

The standard functions (EAX=0 and EAX=1) are the same for all processors. Having standard functions simplifies software's task of testing for and implementing features common to x86 processors. Software can test for these features and, as new x86 processors are released, benefit from these capabilities immediately.

Extended functions are specific to a vendor's processor. These functions provide additional information about AMD processors that software can use to identify enhanced features and functions. To test for extended functions, software checks for a value of at least 8000_0001h in the EAX register returned by function 8000_0000h.

Within AMD's family of processors, different members can execute a different number of functions. Table 1 on page 6 summarizes the CPUID functions currently implemented on AMD processors.

Table 1. Summary of CPUID Functions in AMD Processors

Standard Function	Extended Function	Description	AMD-K5™ Processor (Model 0), Am486®DX4 and Am5$_X$86® Processors	AMD-K5 Processor (Model 1, 2, and 3)	AMD-K6® Processor (Models 6, 7) AMD-K6-2 Processor (Model 8)	AMD-K6-III Processor (Model 9)	AMD Athlon™ Processor (All)
0	—	Vendor String and Largest Standard Function Value	X	X	X	X	X
1	—	Processor Signature and Standard Feature Bits	X	X	X	X	X
—	8000_0000h	Largest Extended Function Value	—	X	X	X	X
—	8000_0001h	Extended Processor Signature and Extended Feature Bits	—	X	X	X	X
—	8000_0002h	Processor Name	—	X	X	X	X
—	8000_0003h	Processor Name	—	X	X	X	X
—	8000_0004h	Processor Name	—	X	X	X	X
—	8000_0005h	L1 TLB*/Cache Information	—	X	X	X	X
—	8000_0006h	L2 TLB/Cache Information	—	—	—	X	X

Notes:

Future versions of these processors may implement additional functions.

Appendix A, "CPUID Instruction Definition" on page 27 contains detailed descriptions of the functions.

* TLB = translation lookaside buffer.

Identifying the Processor Vendor

Software must execute the standard function EAX=0. The CPUID instruction returns a 12-character string that identifies the processor's vendor. The instruction also returns the largest standard function input value defined for the CPUID instruction on the processor.

For AMD processors, function 0 returns a vendor string of "AuthenticAMD". This string informs the software to follow AMD's definition for subsequent CPUID functions and the registers returned for those functions.

Once the software identifies the processor's vendor, it knows the definition for all the functions supplied by the CPUID instruction. By using these functions, the software obtains the

processor information needed to properly tune its functionality to the capabilities of the processor.

Testing For Extended Functions

Software must test for extended functions with function 8000_0000h. The EAX register returns the largest extended function input value defined for the CPUID instruction on the processor. If this value is at least 8000_0001h, extended functions are supported.

With one exception, the AMD extended feature flags include all the information provided in the standard feature flags as well as indicators for the additional AMD processor-specific feature enhancements. The duplication of standard feature bits within the extended feature bits can minimize the number of function calls required by software. The exception is bit 11, which indicates that the SYSENTER and SYSEXIT instructions are supported in the standard features and that the SYSCALL and SYSRET instructions are supported in the extended features.

Determining the Processor Signature

Standard function 1 (EAX=1) of the CPUID instruction returns the standard processor signature and feature bits. The standard processor signature is returned in the EAX register and provides information regarding the specific revision (stepping) and model of the processor and the instruction family level supported by the processor. The revision level can be used to determine if the processor supports specific features. However, it is not recommended that the revision level be used in this manner unless this information is not available through the standard or extended feature bits.

All AMD-K6 processor models belong to instruction family 5 (as returned in EAX by function 1). All AMD Athlon processor models belong to instruction family 6 (as returned in EAX by function 1). Figure 1 on page 8 shows the contents of the EAX register obtained by function 1. Table 2 on page 8 summarizes the specific processor signature values returned for AMD processors.

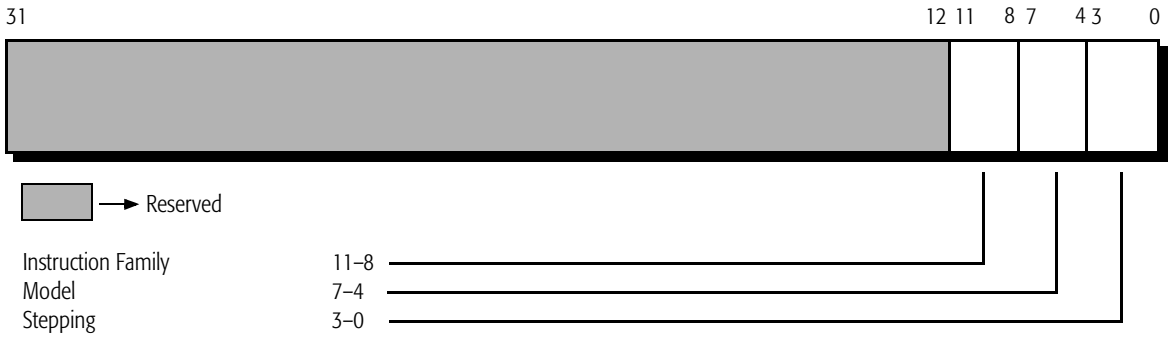

Figure 1. Contents of EAX Register Returned by Function 1

Table 2. Summary of Processor Signatures for AMD Processors

Processor	Instruction Family [11:8]	Model[1] [7:4]	Stepping ID[2] [3:0]
Am486® and Am5$_X$86® Processors	0100b (4h)	yyyy	xxxx
AMD-K5™ Processor (Model 0)	0101b (5h)	0000b (0h)	xxxx
AMD-K5 Processor (Model 1)	0101b (5h)	0001b (1h)	xxxx
AMD-K5 Processor (Model 2)	0101b (5h)	0010b (2h)	xxxx
AMD-K5 Processor (Model 3)	0101b (5h)	0011b (3h)	xxxx
AMD-K6® Processor (Model 6)	0101b (5h)	0110b (6h)	xxxx
AMD-K6 Processor (Model 7)	0101b (5h)	0111b (7h)	xxxx
AMD-K6-2 Processor (Model 8)	0101b (5h)	1000b (8h)	xxxx
AMD-K6-III Processor (Model 9)	0101b (5h)	1001b (9h)	xxxx
AMD Athlon™ Processor (Model 1)	0110b (6h)	0001b (1h)	xxxx
AMD Athlon™ Processor (Model 2)	0110b (6h)	0010b (2h)	xxxx
AMD Athlon™ Processor (Model 4)	0110b (6h)	0100b (4h)	xxxx

Notes:

1. *Contact your AMD representative for Model identifier information.*
2. *Contact your AMD representative for the latest stepping information.*

Identifying Supported Features

The feature bits are returned in the EDX register for two CPUID functions—standard function 1 and extended function 8000_0001h. Each bit corresponds to a specific feature and indicates if that feature is present on the processor. Table 3 summarizes the standard and extended feature bits.

Table 3. Summary of Standard and Extended Feature Bits

Bit[1]	Feature	Description	Standard[2]	Extended[2]
0	Floating-Point Unit	A floating-point unit is available.	1	1
1	Virtual Mode Extensions	Virtual mode extensions are available.	1	1
2	Debugging Extensions	I/O breakpoint debug extensions are supported.	1	1
3	PSE (Page Size Extensions)	4-Mbyte pages are supported.	1	1
4	Time Stamp Counter (with RDTSC and CR4 disable bit)	A time stamp counter is available in the processor, and the RDTSC instruction is supported.	1	1
5	K86™ Family of Processors' Model-Specific Registers (with RDMSR and WRMSR)	The K86 model-specific registers are available in the processor, and the RDMSR and WRMSR instructions are supported.	1	1
6	PAE (Page Address Extensions)	Page address extensions are supported using an 8-byte directory entry.	1	1
7	MCE (Machine Check Exception)	The machine check exception is supported.	1	1
8	CMPXCHG8B Instruction	The CMPXCHG8B instruction is supported.	1	1
9	APIC	A local APIC unit is available.	0[3]	0[3]
11	SYSENTER/SYSEXIT Instructions	The SYSENTER and SYSEXIT instructions are supported.	1	0
	SYSCALL and SYSRET Instructions	The SYSCALL and SYSRET instructions and associated extensions are supported.	0	1
12	MTRR (Memory Type Range Registers)	Memory type range registers are available.	1	1
13	Global Paging Extension	Global paging extensions are available.	1	1
14	MCA (Machine Check Architecture)	Machine check architecture is supported	1	1
15	Conditional Move Instructions	The conditional move instructions, CMOV and FCMOV, are supported. The FCOMI instruction is also supported.	1	1
16	PAT (Page Attribute Table)	The Page attribute tables are supported.	1	1

Note:

1. *Appendix A, "CPUID Instruction Definition" on page 27 contains details on bit locations and values.*
2. *Bit definitions: 0 = No Support, 1 = Support*
3. *BIOS must enable the local APIC.*

Table 3. Summary of Standard and Extended Feature Bits (continued)

Bit[1]	Feature	Description	Standard[2]	Extended[2]
17	PSE-36 (Page Size Extension)	Page size extensions for 36-bit addresses are supported using a 4-byte directory entry.	1	1
22	AMD Multimedia Instruction Extensions	AMD additions to the original MMX™ instruction set are supported.	0	1
23	MMX™ Instructions	The MMX instruction set is supported.	1	1
24	FXSAVE/FXRSTOR	Fast floating-point save and restore is supported.	1	1
30	AMD 3DNow!™ Instruction Extensions	Extensions to the 3DNow! instruction set are supported.	0	1
31	3DNow! Instructions	3DNow! instructions are supported.	0	1

Note:

1. *Appendix A, "CPUID Instruction Definition" on page 27 contains details on bit locations and values.*
2. *Bit definitions: 0 = No Support, 1 = Support*
3. *BIOS must enable the local APIC.*

Before using any of the enhanced features added to the latest generation of processors, software should test each feature bit returned by functions 1 and 8000_0001h to identify the capabilities available on the processor. For example, software must test feature bit 23 to determine if the processor executes the MMX technology instructions. Attempting to execute an unavailable feature can cause errors and exceptions.

Bit 31, as returned by extended function 8000_0001h, designates the presence of 3DNow! technology. Other processor vendors have adopted this technology, so bit 31 is now considered an open standard. Appendix A, "CPUID Instruction Definition" on page 27 and Appendix B, "Register Values Returned by the AMD Family Processors" on page 41 contain details on bit locations and values.

Determining Instruction Set Support

It is preferable to use CPUID feature flags as much as possible, rather than deriving capabilities from vendor specifiers combined with CPUID model numbers.

The AMD Athlon processor has two new sets of powerful extensions to the x86 instruction set—3DNow! instruction extensions and multimedia enhancement extensions. See the *AMD Additions to the 3DNow!™ and MMX™ Instruction Sets Manual*, order# 22466 for more information about these 24 new instructions.

To simplify the detection of the new instructions and the original 3DNow! and MMX instructions, use the following algorithm. A code sample using the CPUID instruction to identify the processor and its features is available from AMD's website at http://www.amd.com/products/cpg/bin/cpuid_ex.zip. There are other ways to implement detection besides the way shown in the sample.

CPUID Test

1. Establish that the processor has support for CPUID. See "Testing for the CPUID Instruction" on page 4.

Standard Function Test

2. Execute CPUID function 0, which returns the processor vendor string and the highest standard function supported. Save the vendor string for a later comparison. (See step 9.)
3. If step 2 indicates that the highest standard function is at least 1, execute CPUID function 1, which returns the standard feature flags in the EDX register.

MMX Test

4. If bit 23 of the standard feature flags is set to 1, MMX technology is supported. MMX instruction support is the basic minimum processor feature required to support other instruction extensions.

Optional SSE Test

5. Optionally, if bit 25 of the standard feature flags is set, the processor has streaming SIMD extensions (SSE) capabilities. Further qualification of SSE is done by checking for OS support. SSE support might be present in the processor, but not usable due to a lack of OS support for the additional architected registers.

Extended Functions Test

6. Execute CPUID extended function 8000_0000h. This function returns the highest extended function supported in EAX. If EAX=0, there is no support for extended functions.
7. If the highest extended function supported is at least 8000_0001h, execute CPUID function 8000_0001h. This function returns the extended feature flags in EDX.

3DNow! Instruction Test

8. If bit 31 of the extended feature flags is set to 1, the 3DNow! instructions are supported.

Vendor Check

9. If the previously saved vendor string (see step 2) contains "AuthenticAMD", continue on to the next step.

3DNow! Extensions Test

10. If bit 30 of the extended feature flags is set to 1, the additions to the 3DNow! instruction set are supported.

MMX Extensions Test

11. If bit 22 of the extended feature flags is set to 1, the new multimedia enhancement instructions that augment the MMX instruction set are supported.

AMD Processor Signature (Extended Function)

Extended function 8000_0001h returns the AMD processor signature. The signature is returned in the EAX register and provides generation, model, and stepping information for AMD processors. Figure 2 on page 12 shows the contents returned in the EAX register.

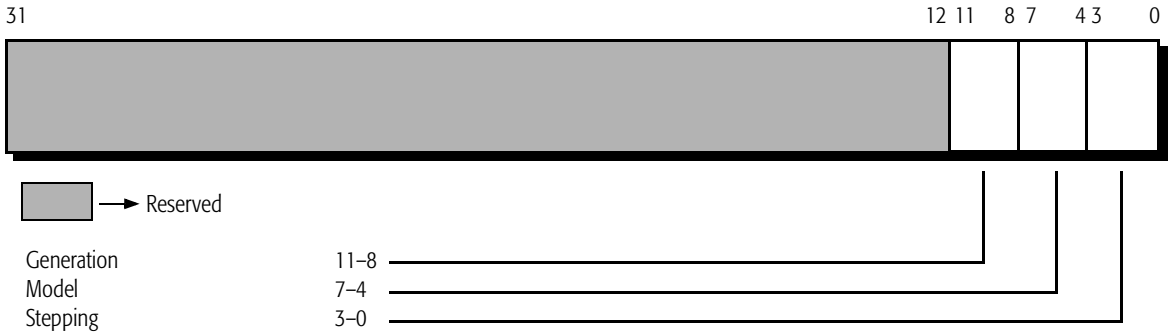

Figure 2. Contents of EAX Register Returned by Extended Function 8000_0001h

Displaying the Processor Name

Functions 8000_0002h, 8000_0003h, and 8000_0004h return an ASCII string containing the name of the processor. These functions eliminate the need for software to search for the processor name in a lookup table, a process requiring a large block of memory and frequent updates. Instead, software can simply call these three functions to obtain the name string (48 ASCII characters in little endian format) and display it on the screen. Although the name string can be up to 48 characters in length, shorter names have the remaining byte locations filled with the ASCII NULL character (00h). To simplify the display routines and avoid using screen space, software only needs to display characters until a NULL character is detected.

Note: *The AMD Athlon™ processor (Models 2 and 4) returns the processor name string AMD Athlon(tm) Processor. This*

name string is different that the one returned for the AMD Athlon processor (Model 1).

Table 4 summarizes the name strings returned for the AMD Athlon processor Models 1, 2, and 4.

Table 4. Processor Name Strings for the AMD Athlon™ Processor

Processor	Name String
AMD Athlon™ Processor (Model 1)	AMD-K7(tm) Processor
AMD Athlon™ Processor (Model 2)	AMD Athlon(tm) Processor
AMD Athlon™ Processor (Model 4)	AMD Athlon(tm) Processor

Displaying Cache Information

Functions 8000_0005h and 8000_0006h provide cache information for the processor, although function 8000_0006h is only supported on the AMD Athlon processor, and the AMD-K6®-III processor Model 9. Some diagnostic software displays information about the system and the processor's configuration. It is common for this type of software to provide cache size and organization of information.

Functions 8000_0005h and 8000_0006h provide a simple way for software to obtain information about the on-chip cache and translation lookaside buffer (TLB) structures. The size and organization information is returned in the registers as described in Appendix A on page 27. Software can simply display these values, eliminating the need for large pieces of code to test the memory structures.

Code Samples

Developers who want to create their own processor-features detection code should follow the sample code described in "Example CPUID Code".

A more elaborate function call, which detects the full range of CPUID information, is provided as sample code in "Example Function Call" on page 22.

Example CPUID Code

Developers who want to create their own processor detection program should follow the algorithm in the “cpuid_ex” program. The code sample is available from AMD’s website at the following url:

http://www.amd.com/products/cpg/bin/cpuid_ex.zip

The source code is included, along with an executable that was compiled with Microsoft® Visual Studio C/C++ Versions 5 and 6. This example provides a simple algorithm for the developer to follow and can be accommodated by many different processors. The source code, cpuid_ex.c, follows the recommendations described in this document.

To display a list of supported features for the processor, run the program by typing

```
cpuid_ex
```

in a Microsoft Windows® 95, Windows® 98, or Windows NT® 4.0 DOS window.

For convenience, the example CPUID code is displayed below.

```
/* The following code follows the guidelines described in this document,
It is meant to serve as only an example, as there are other ways to accomplish
processor detection. */

#include <stdio.h>
#include <excpt.h>

/* Symbolic constants for feature flags in CPUID standard feature flags */

#define CPUID_STD_FPU          0x00000001
#define CPUID_STD_VME          0x00000002
#define CPUID_STD_DEBUGEXT     0x00000004
#define CPUID_STD_4MPAGE       0x00000008
#define CPUID_STD_TSC          0x00000010
#define CPUID_STD_MSR          0x00000020
#define CPUID_STD_PAE          0x00000040
#define CPUID_STD_MCHKXCP      0x00000080
#define CPUID_STD_CMPXCHG8B    0x00000100
#define CPUID_STD_APIC         0x00000200
#define CPUID_STD_SYSENTER     0x00000800
#define CPUID_STD_MTRR         0x00001000
#define CPUID_STD_GPE          0x00002000
#define CPUID_STD_MCHKARCH     0x00004000
#define CPUID_STD_CMOV         0x00008000
#define CPUID_STD_PAT          0x00010000
#define CPUID_STD_PSE36        0x00020000
#define CPUID_STD_MMX          0x00800000
#define CPUID_STD_FXSAVE       0x01000000
#define CPUID_STD_SSE          0x02000000


/* Symbolic constants for feature flags in CPUID extended feature flags */

#define CPUID_EXT_3DNOW        0x80000000
#define CPUID_EXT_AMD_3DNOWEXT 0x40000000
#define CPUID_EXT_AMD_MMXEXT   0x00400000


/* Symbolic constants for application specific feature flags */

#define FEATURE_CPUID          0x00000001
#define FEATURE_STD_FEATURES   0x00000002
#define FEATURE_EXT_FEATURES   0x00000004
#define FEATURE_TSC            0x00000010
#define FEATURE_MMX            0x00000020
#define FEATURE_CMOV           0x00000040
#define FEATURE_3DNOW          0x00000080
#define FEATURE_3DNOWEXT       0x00000100
#define FEATURE_MMXEXT         0x00000200
#define FEATURE_SSEFP          0x00000400
#define FEATURE_K6_MTRR        0x00000800
#define FEATURE_P6_MTRR        0x00001000
```

```
/* Older compilers do not support the CPUID instruction in inline assembly */

#define cpuid _asm _emit 0x0f _asm _emit 0xa2


/* get_feature_flags extracts all features the application wants to know
   about from CPUID information and returns a bit string of application
   specific feature bits. The following design criteria apply:

   1. Processor capabilities should be directly derived from CPUID feature bits
      wherever possible, instead of being derived from vendor strings and
      processor signatures. However, some features are not indicated by CPUID
      feature flags (whether basic or extended) and do require looking at
      vendor strings and processor signatures. Applications may also choose to
      implement pseudo capabilities, for example indicating performance
      levels.
   2. The basic feature flags returned by CPUID function #1 are compatible
      across all x86 processor vendors with very few exceptions and therefore
      common feature checks for things like MMX or TSC support do not require
      a vendor check before evaluating the basic feature flag information.
      If unsure about a particular feature, review the processor vendor's
      literature.
   3. 3DNow! technology is an open standard. Therefore 3DNow! instruction
      capabilities are indicated by bit 31 in the extended feature flags
      regardless of processor vendor.
   4. Applications should always treat the floating-point part of SSE and
      the MMX part of SSE as separate capabilities because SSE FP requires
      OS support that might not be available, while SSE MMX works with all
      operating systems.
*/


unsigned int get_feature_flags(void)
{
   unsigned int result    = 0;
   unsigned int signature = 0;
      char vendor[13]       = "UnknownVendr";  /* Needs to be exactly 12 chars */

   /* Define known vendor strings here */

   char vendorAMD[13]     = "AuthenticAMD";  /* Needs to be exactly 12 chars */
      /*;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
         ;; Step 1: Check if processor has CPUID support. The processor faults
         ;; with an illegal instruction exception if the instruction is not
         ;; supported. This step catches the exception and immediately returns
         ;; with feature string bits with all 0s, if the exception occurs.
         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;*/
      __try {
            __asm xor    eax, eax
            __asm xor    ebx, ebx
```

```
        __asm xor    ecx, ecx
        __asm xor    edx, edx
        __asm cpuid
    }

    __except (EXCEPTION_EXECUTE_HANDLER) {
        return (0);
    }

    result |= FEATURE_CPUID;

    _asm {

       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
       ;; Step 2: Check if CPUID supports function 1 (signature/std features)
       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

       xor     eax, eax                      ; CPUID function #0
       cpuid                                 ; largest std func/vendor string
       mov     dword ptr [vendor], ebx       ; save
       mov     dword ptr [vendor+4], edx     ;  vendor
       mov     dword ptr [vendor+8], ecx     ;   string
       test    eax, eax                      ; largest standard function==0?
       jz      $no_standard_features         ; yes, no standard features func
       or      [result], FEATURE_STD_FEATURES; does have standard features

       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
       ;; Step 3: Get standard feature flags and signature
       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

       mov     eax, 1                        ; CPUID function #1
       cpuid                                 ; get signature/std feature flgs
       mov     [signature], eax              ; save processor signature

       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
       ;; Step 4: Extract desired features from standard feature flags
       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

       ;; Check for time stamp counter support

       mov     ecx, CPUID_STD_TSC            ; bit 4 indicates TSC support
       and     ecx, edx                      ; supports TSC ? CPUID_STD_TSC:0
       neg     ecx                           ; supports TSC ? CY : NC
       sbb     ecx, ecx                      ; supports TSC ? 0xffffffff:0
       and     ecx, FEATURE_TSC              ; supports TSC ? FEATURE_TSC:0
       or      [result], ecx                 ; merge into feature flags

       ;; Check for MMX support

       mov     ecx, CPUID_STD_MMX            ; bit 23 indicates MMX support
       and     ecx, edx                      ; supports MMX ? CPUID_STD_MMX:0
```

```
neg     ecx                             ; supports MMX ? CY : NC
sbb     ecx, ecx                        ; supports MMX ? 0xffffffff:0
and     ecx, FEATURE_MMX                ; supports MMX ? FEATURE_MMX:0
or      [result], ecx                   ; merge into feature flags

;; Check for CMOV support

mov     ecx, CPUID_STD_CMOV             ; bit 15 indicates CMOV support
and     ecx, edx                        ; supports CMOV?CPUID_STD_CMOV:0
neg     ecx                             ; supports CMOV ? CY : NC
sbb     ecx, ecx                        ; supports CMOV ? 0xffffffff:0
and     ecx, FEATURE_CMOV               ; supports CMOV ? FEATURE_CMOV:0
or      [result], ecx                   ; merge into feature flags

;; Check support for P6-style MTRRs

mov     ecx, CPUID_STD_MTRR             ; bit 12 indicates MTRR support
and     ecx, edx                        ; supports MTRR?CPUID_STD_MTRR:0
neg     ecx                             ; supports MTRR ? CY : NC
sbb     ecx, ecx                        ; supports MTRR ? 0xffffffff:0
and     ecx, FEATURE_P6_MTRR            ; supports MTRR ? FEATURE_MTRR:0
or      [result], ecx                   ; merge into feature flags

;; Check for initial SSE support. There can still be partial SSE
;; support. Step 9 will check for partial support.

mov     ecx, CPUID_STD_SSE              ; bit 25 indicates SSE support
and     ecx, edx                        ; supports SSE ? CPUID_STD_SSE:0
neg     ecx                             ; supports SSE ? CY : NC
sbb     ecx, ecx                        ; supports SSE ? 0xffffffff:0
and     ecx, (FEATURE_MMXEXT+FEATURE_SSEFP) ; supports SSE ?
                                        ; FEATURE_MMXEXT+FEATURE_SSEFP:0
or      [result], ecx                   ; merge into feature flags

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Step 5: Check for CPUID extended functions
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

mov     eax, 0x80000000                 ; extended function 0x80000000
cpuid                                   ; largest extended function
cmp     eax, 0x80000000                 ; no function > 0x80000000 ?
jbe     $no_extended_features           ; yes, no extended feature flags
or      [result], FEATURE_EXT_FEATURES; does have ext. feature flags

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Step 6: Get extended feature flags
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

mov     eax, 0x80000001                 ; CPUID ext. function 0x80000001
cpuid                                   ; EDX = extended feature flags
```

```
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;; Step 7: Extract vendor independent features from extended flags
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    ;; Check for 3DNow! instruction support (vendor independent)

    mov     ecx, CPUID_EXT_3DNOW          ; bit 31 indicates 3DNow! support
    and     ecx, edx                    ; supports 3DNow! ?CPUID_EXT_3DNOW:0
    neg     ecx                           ; supports 3DNow! ? CY : NC
    sbb     ecx, ecx                      ; supports 3DNow! ? 0xffffffff:0
    and     ecx, FEATURE_3DNOW            ; support 3DNow!?FEATURE_3DNOW:0
    or      [result], ecx                 ; merge into feature flags

    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;; Step 8: Determine CPU vendor
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    lea     esi, vendorAMD                ; AMD's vendor string
    lea     edi, vendor                   ; this CPU's vendor string
    mov     ecx, 12                       ; strings are 12 characters
    cld                                   ; compare lowest to highest
    repe    cmpsb                         ; current vendor string == AMD's ?
    jnz     $not_AMD                      ; no, CPU vendor is not AMD

    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;; Step 9: Check AMD specific extended features
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    mov     ecx, CPUID_EXT_AMD_3DNOWEXT   ; bit 30 indicates 3DNow! ext.
    and     ecx, edx                      ; 3DNow! ext?
    neg     ecx                           ; 3DNow! ext ? CY : NC
    sbb     ecx, ecx                      ; 3DNow! ext ? 0xffffffff : 0
    and     ecx, FEATURE_3DNOWEXT         ; 3DNow! ext?FEATURE_3DNOWEXT:0
    or      [result], ecx                 ; merge into feature flags

    test    [result], FEATURE_MMXEXT      ; determined SSE MMX support?
    jnz     $has_mmxext                   ; yes, don't need to check again

    ;; Check support for AMD's multimedia instruction set additions

    mov     ecx, CPUID_EXT_AMD_MMXEXT     ; bit 22 indicates MMX extension
    and     ecx, edx                      ; MMX ext?CPUID_EXT_AMD_MMXEXT:0
    neg     ecx                           ; MMX ext? CY : NC
    sbb     ecx, ecx                      ; MMX ext? 0xffffffff : 0
    and     ecx, FEATURE_MMXEXT           ; MMX ext ? FEATURE_MMXEXT:0
    or      [result], ecx                 ; merge into feature flags

$has_mmxext:
```

```
      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
      ;; Step 10: Check AMD-specific features not reported by CPUID
      ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

      ;; Check support for AMD-K6 processor-style MTRRs

      mov     eax, [signature] ; get processor signature
      and     eax, 0xFFF ; extract family/model/stepping
      cmp     eax, 0x588 ; CPU < AMD-K6-2/CXT ? CY : NC
      sbb     edx, edx ; CPU < AMD-K6-2/CXT ? 0xffffffff:0
      not     edx ; CPU < AMD-K6-2/CXT ? 0:0xffffffff
      cmp     eax, 0x600 ; CPU < AMD Athlon ? CY : NC
      sbb     ecx, ecx ; CPU < AMD-K6 ? 0xffffffff:0
      and     ecx, edx ; (CPU>=AMD-K6-2/CXT)&&
                       ; (CPU<AMD Athlon) ? 0xffffffff:0
      and     ecx, FEATURE_K6_MTRR ; (CPU>=AMD-K6-2/CXT)&&
                       ; (CPU<AMD Athlon) ? FEATURE_K6_MTRR:0
      or      [result], ecx ; merge into feature flags

      jmp     $all_done ; desired features determined

   $not_AMD:

      /* Extract features specific to non AMD CPUs */

   $no_extended_features:
   $no_standard_features:
   $all_done:
}

/* The FP part of SSE introduces a new architectural state and therefore
   requires support from the operating system. So even if CPUID indicates
   support for SSE FP, the application might not be able to use it. If
   CPUID indicates support for SSE FP, check here whether it is also
   supported by the OS, and turn off the SSE FP feature bit if there
   is no OS support for SSE FP.

   Operating systems that do not support SSE FP return an illegal
   instruction exception if execution of an SSE FP instruction is performed.
   Here, a sample SSE FP instruction is executed, and is checked for an
   exception using the (non-standard) __try/__except mechanism
   of Microsoft Visual C.
*/

if (result & FEATURE_SSEFP) {
    __try {
       __asm _emit 0x0f
       __asm _emit 0x56
       __asm _emit 0xC0    ;; orps xmm0, xmm0
       return (result);
    }
```

```
        __except (EXCEPTION_EXECUTE_HANDLER) {
           return (result & (~FEATURE_SSEFP));
        }
   }
   else {
      return (result);
   }
}



/* The  sample "application" */

int main (void)
{
   unsigned int capabilities = get_feature_flags();

   printf ("features = %08x\n", capabilities);
   printf ("CPU supports CPUID:       %c\n",
           capabilities & FEATURE_CPUID ? 'y' : 'n');
   printf ("CPU supports CPUID STD:   %c\n",
           capabilities & FEATURE_STD_FEATURES ? 'y' : 'n');
   printf ("CPU supports CPUID EXT:   %c\n",
           capabilities & FEATURE_EXT_FEATURES ? 'y' : 'n');
   printf ("CPU supports TSC:         %c\n",
           capabilities & FEATURE_TSC ? 'y' : 'n');
   printf ("CPU supports CMOV:        %c\n",
           capabilities & FEATURE_CMOV ? 'y' : 'n');
   printf ("CPU supports MMX:         %c\n",
           capabilities & FEATURE_MMX ? 'y' : 'n');
   printf ("CPU supports 3DNOW:       %c\n",
           capabilities & FEATURE_3DNOW ? 'y' : 'n');
   printf ("CPU supports 3DNOW_EXT:   %c\n",
           capabilities & FEATURE_3DNOWEXT ? 'y' : 'n');
   printf ("CPU supports AMD-K6-MTRR: %c\n",
           capabilities & FEATURE_K6_MTRR ? 'y' : 'n');
   printf ("CPU supports P6-MTRR:     %c\n",
           capabilities & FEATURE_P6_MTRR ? 'y' : 'n');
   printf ("CPU supports SSE MMX:     %c\n",
           capabilities & FEATURE_MMXEXT ? 'y' : 'n');
   printf ("CPU supports SSE FPU:     %c\n",
           capabilities & FEATURE_SSEFP ? 'y' : 'n');
   return (0);
}
```

Example Function Call

This code sample detects the full range of CPUID information and allows the user to query capabilities through a simple function call. The code sample is available from AMD's website at the following url:

http://www.amd.com/products/cpg/bin/getcpu_caps.zip

The zip file contains two files—DETECT.C and ADETECT.H. Follow these steps to use the function call:

1. Copy DETECT.C and ADETECT.H into your project directory
2. Add DETECT.C to your source project

Now the user can make calls to GetCPUCaps() in any module that includes ADETECT.H. Add the function call with the following statement:

```
#include "ADETECT.H"
```

This source code compiles under Microsoft Visual Studio C/C++ Versions 5 and 6.

Displaying the AMD Athlon™ Processor Name String

All AMD Athlon processors support CPUID extended functions 8000_0002h, 8000_0003h, and 8000_0004h. These functions return an ASCII string containing the name of the processor. These functions eliminate the need for software to search for the processor name in a look-up table. Instead, software can simply call these three functions to obtain the name string (up to 48 ASCII characters in little-endian format) and display it on the screen. The character string is terminated with a 00h (ASCII null character).

The following code samples illustrate methods that can be used to display the processor name string as required by the AMD Athlon processor branding strategy.

DisplayK7NameString Subroutine

The AMD Athlon processor name string can be displayed by calling the subroutine DisplayK7NameString. The following code sample displays the processor name string.

```
;---------------------------------------------------------------------------
; DisplayK7NameString:
;
;
; Returns:
;     cf=0 all 48 possible characters displayed
;     cf=1 end of string reached
;---------------------------------------------------------------------------
DisplayK7NameString proc near ;
      push  eax               ;
      push  ebx               ;
      push  ecx               ;
      push  edx               ;
      K7_CPUID 80000002h      ;
      call  DisplayK7NameSubstring;
      jc    @f                ;End of string?
      K7_CPUID 80000003h      ;
      call  DisplayK7NameSubstring;
      jc    @f                ;End of string?
      K7_CPUID 80000004h      ;
      call  DisplayK7NameSubstring;
@@:   pop   edx               ;
      pop   ecx               ;
      pop   ebx               ;
      pop   eax               ;
      ret                     ;
DisplayK7NameString endp      ;
;---------------------------------------------------------------------------
```

K7_CPUID

K7_CPUID is an AMD macro that generates a CPUID instruction and, optionally, loads the EAX register with the specified function number.

```
K7_CPUID macro cpuidindex
   IFNB <cpuidindex>
      mov   eax, cpuidindex
   ENDIF
      Db    0Fh, 0A2h   ;CPUID instruction
      endm
```

Using K7CPUID, the following line of code:

```
K7_CPUID 80000002h
```

generates the following instructions:

```
mov eax, 80000002h
CPUID
```

DisplayK7NameSubstring

The DisplayK7NameSubstring subroutine is called up to three times to display the ASCII characters returned by each CPUID function call.

```
;-----------------------------------------------------------------------
; DisplayK7NameSubstring:
;
;
; Returns:
;     cf=0 no errors
;     cf=1 end of string reached
;-----------------------------------------------------------------------
DisplayK7NameSubstring proc near;Displays eax, ebx, ecx, edx
      call  DisplayEaxAscii   ; eax
      jc    @f                ;End of string?
      xchg  eax, ebx          ;
      call  DisplayEaxAscii   ; ebx
      jc    @f                ;End of string?
      xchg  eax, ecx          ;
      call  DisplayEaxAscii   ; ecx
      jc    @f                ;End of string?
      xchg  eax, edx          ;
      call  DisplayEaxAscii   ; edx
@@:   ret                     ;
DisplayK7NameSubstring endp   ;
;-----------------------------------------------------------------------
```

DisplayEaxAscii

The DisplayK7NameSubstring subroutine calls the DisplayEaxAscii subroutine up to four times. DisplayEaxAscii displays the four bytes of the EAX register as ASCII characters starting with the least-significant byte (little endian). The subroutine DisplayAlChar used in the example is a generic name for a subroutine that displays the value in the AL register as an ASCII character. This type of subroutine is common to all BIOS under a variety of names.

```
;-----------------------------------------------------------------------
; DisplayEaxAscii:
;
; Returns:
;     cf=0 no errors
;     cf=1 end of string reached
;-----------------------------------------------------------------------
DisplayEaxAscii proc near     ;
      push  eax               ;
      push  cx                ;
      mov   cx, 4             ;
;---------------------------------------;
@@:   or    al, al            ;End of string?
      stc                     ;(assume end of string)
      jz    @f                ; YES--assumed correctly
      call  DisplayAlChar     ; NO---display character
      ror   eax, 8            ;next char in al
      loop  @b                ;repeat
;---------------------------------------;
      clc                     ;
@@:   pop   cx                ;Restore regs
      pop   eax               ;
      ret                     ;
DisplayEaxAscii endp          ;
;-----------------------------------------------------------------------
```

Appendix A

CPUID Instruction Definition

This appendix contains a detailed description of the CPUID instruction.

CPUID Instruction

mnemonic	*opcode*	*description*
CPUID	0F A2h	Identify the processor and its feature set
Privilege:	none	
Registers Affected:	EAX, EBX, ECX, EDX	
Flags Affected:	none	
Exceptions Generated:	none	

The CPUID instruction is an application-level instruction that software executes to identify the processor and its feature set. This instruction offers multiple functions, each providing a different set of information about the processor. The CPUID instruction can be executed from any privilege level. Software can use the information returned by this instruction to tune its functionality for the specific processor and its features.

Not all processors implement the CPUID instruction. Therefore, software must test to determine if the instruction is present on the processor. If the ID bit (21) in the EFLAGS register is writeable, the CPUID instruction is implemented.

The CPUID instruction supports multiple functions. The information associated with each function is obtained by executing the CPUID instruction with the function number in the EAX register. Functions are divided into two types: standard functions and extended functions. Standard functions are found in the low function space, 0000_0000h–7FFF_FFFFh. In general, all x86 processors have the same standard function definitions.

Extended functions are defined specifically for processors supplied by the vendor listed in the vendor identification string. Extended functions are found in the high function space, 8000_0000h–8FFF_FFFFh. Because not all vendors have defined extended functions, software must test for their presence on the processor. AMD processors have extended functions if the 8000_0000h function returns a value of at least 8000_0001h in the EAX register.

Standard Functions

Function 0 — Largest Standard Function Input Value and Vendor Identification String

Input: EAX = 0

Output: EAX = Largest function input value recognized by the CPUID instruction
EBX, EDX, ECX = Vendor identification string

This is a standard function found in all processors implementing the CPUID instruction. It returns two values. The first value is returned in the EAX register and indicates the largest standard function value recognized by the processor. The second value is the vendor identification string. This 12-character ASCII string is returned in the EBX, EDX, and ECX registers in little endian format. AMD processors return a vendor identification string of “AuthenticAMD” as follows:

EBX				**EDX**				**ECX**				← Registers
h	t	u	A	i	t	n	e	D	M	A	c	← Alpha Characters
68	74	75	41	69	74	6E	65	44	4D	41	63	← ASCII Codes

Software uses the vendor identification string as follows:

- To identify the processor as an AMD processor
- To apply AMD's definition of the CPUID instruction for all additional function calls

Function 1 — Processor Signature and Standard Feature Flags

Input: EAX = 1

Output: EAX = Processor Signature
EBX = Reserved
ECX = Reserved
EDX = Standard Feature Flags

Function 1 returns two values—the Processor Signature and the Standard Feature Flags. The processor signature is returned in the EAX register and identifies the specific processor by providing information on its type—instruction family, model, and revision (stepping). The information is formatted as follows:

- EAX[3–0] Stepping ID
- EAX[7–4] Model
- EAX[11–8] Instruction Family
- EAX[31–12] Reserved

The standard feature flags are returned in the EDX register and indicate the presence of specific features. In most cases, a "1" indicates the feature is present, and a "0" indicates the feature is not present. Table 5 on page 30 contains a list of the currently defined standard feature flags for the AMD-K6® processor Models 8 and 9. Table 6 on page 31 contains a list of the currently defined standard feature flags for the AMD Athlon™ processor Models 1, 2 and 4. (See tables 21 through 23 in Appendix B, "Register Values Returned by the AMD Family Processors" on page 41 for all K86 family processor register definitions, including the AMD-K6 processor Models 6 and 7.) Reserved bits will be used for new features as they are added. For more information, see "The AMD Athlon™ Processor Model 1, Model 2, and Model 4" on page 2.

Table 5. Standard Feature Flag Descriptions for the AMD-K6®-2 and AMD-K6®-III Processors

Bit	Feature[1]	AMD-K6®-2 Processor (Model 8)	AMD-K6-III Processor (Model 9)
0	Floating-Point Unit	1	1
1	Virtual Mode Extensions	1	1
2	Debugging Extensions	1	1
3	Page Size Extensions (4-Mbyte pages)	1	1
4	Time Stamp Counter (with RDTSC and CR4 disable bit)	1	1
5	K86™ Family of Processors' Model-Specific Registers (with RDMSR and WRMSR)	1	1
6	PAE (Page Address Extensions)	0	0
7	Machine Check Exception	1	1
8	CMPXCHG8B Instruction	1	1
9	APIC	0	0
10	*Reserved on all AMD processors*	0	0
11	SYSENTER/SYSEXIT[2]	0	0
12	Memory Type Range Registers	0	0
13	Global Paging Extension	1[3]	1
14	Machine Check Architecture	0	0
15	Conditional Move Instruction	0	0
16	PAT (Page Attribute Table)	0	0
17	PSE-36 (Page Size Extensions)	0	0
18–22	*Reserved on all AMD processors*	0	0
23	MMX™ Instructions	1	1
24	FXSAVE/FXRSTOR	0	0
25–31	*Reserved on all AMD processors*	0	0

Notes:

1. *Bit definitions: 0 = No Support, 1 = Support.*
2. *The SYSENTER and SYSEXIT instructions have different implementations than the SYSCALL and SYSRET instructions.*
3. *See Table 22 on page 44 for more information about Global Paging Extensions in the AMD-K6-2 processor Model 8.*

Table 6. Standard Feature Flag Descriptions for the AMD Athlon™ Processors

Bit	Feature[1]	AMD Athlon™ Processor (Model 1)	AMD Athlon™ Processor (Model 2)	AMD Athlon™ Processor (Model 4)
0	Floating-Point Unit	1	1	1
1	Virtual Mode Extensions	1	1	1
2	Debugging Extensions	1	1	1
3	Page Size Extensions (4-Mbyte pages)	1	1	1
4	Time Stamp Counter (with RDTSC and CR4 disable bit)	1	1	1
5	K86™ Family of Processors' Model-Specific Registers (with RDMSR and WRMSR)	1	1	1
6	PAE (Page Address Extensions)	1	1	1
7	Machine Check Exception	1	1	1
8	CMPXCHG8B Instruction	1	1	1
9	APIC	0	1[2]	1[2]
10	*Reserved on all AMD processors*	0	0	0
11	SYSENTER/SYSEXIT[3]	1	1	1
12	Memory Type Range Registers	1	1	1
13	Global Paging Extension	1	1	1
14	Machine Check Architecture	1	1	1
15	Conditional Move Instruction	1	1	1
16	PAT (Page Attribute Table)	1	1	1
17	PSE-36 (Page Size Extensions)	0	1	1
18–22	*Reserved on all AMD processors*	0	0	0
23	MMX™ Instructions	1	1	1
24	FXSAVE/FXRSTOR	0	1	1
25–31	*Reserved on all AMD processors*	0	0	0

Notes:

1. *Bit definitions: 0 = No Support, 1 = Support.*
2. *The AMD Athlon processor Model 2 and Model 4 contain a local APIC. The BIOS must enable the local APIC in order for bit 9 to return a 1 (supported).*
3. *The SYSENTER and SYSEXIT instructions have different implementations than the SYSCALL and SYSRET instructions.*

Extended Functions

Function 8000_0000h – Largest Extended Function Input Value

Input: EAX = 8000_0000h

Output: EAX = Largest function input value recognized by the CPUID instruction
EBX = Reserved
ECX = Reserved
EDX = Reserved

Function 8000_0000h returns a value in the EAX register that indicates the largest extended function value recognized by the processor.

Function 8000_0001h – AMD Processor Signature and Extended Feature Flags

Input: EAX = 8000_0001h

Output: EAX = AMD Processor Signature
EBX = Reserved
ECX = Reserved
EDX = Extended Feature Flags

Function 8000_0001h returns two values—the AMD Processor Signature and the Extended Feature Flags. The AMD processor signature is returned in the EAX register and identifies the specific processor by providing information regarding its type—generation, model, and revision (stepping). (The instruction family can be obtained by using function 1.) The information for function 8000_0001h is formatted as follows:

- EAX[3–0] Stepping ID
- EAX[7–4] Model
- EAX[11–8] Generation
- EAX[31–12] Reserved

The extended feature flags are returned in the EDX register and indicate the presence of specific features found in AMD processors. In most cases, a ‘1’ indicates the feature is present, and a ‘0’ indicates the feature is not present. Table 7 on page 33 contains a list of the currently defined feature flags for the AMD-K6 processor Models 8 and 9. Table 8 on page 34 contains a list of the currently defined feature flags for the the AMD Athlon processor. (See tables 21 through 23 in Appendix B, "Register Values Returned by the AMD Family Processors" on page 41 for all K86 family processor register definitions.) Reserved bits will be used for new features as they are added.

Table 7. Extended Feature Flag Descriptions for the AMD-K6®-2 and AMD-K6®-III Processors

Bit	Feature[1]	AMD-K6®-2 Processor (Model 8)	AMD-K6-III Processor (Model 9)
0	Floating-Point Unit	1	1
1	Virtual Mode Extensions	1	1
2	Debugging Extensions	1	1
3	Page Size Extensions (4-Mbyte Pages)	1	1
4	Time Stamp Counter (with RDTSC and CR4 disable bit)	1	1
5	K86™ Family of Processors' Model-Specific Registers (with RDMSR and WRMSR)	1	1
6	PAE (Page Address Extensions)	0	0
7	Machine Check Exception	1	1
8	CMPXCHG8B Instruction	1	1
9	APIC	0	0
10	*Reserved on all AMD processors*	0	0
11	SYSCALL and SYSRET Instructions[2]	1	1
12	Memory Type Range Registers	0	0
13	Global Paging Extension	1	1
14	Machine Check Architecture	0	0
15	Conditional Move Instruction	0	0
16	PAT (Page Attribute Table)	0	0
17	PSE-36 (Page Size Extensions)	0	0
18–21	*Reserved on all AMD processors*	0	0
22	AMD MMX™ Instruction Extensions	0	0
23	MMX Instructions	1	1
24	FXSAVE/FXRSTOR	0	0
25–29	*Reserved on all AMD processors*	0	0
30	AMD 3DNow!™ Instruction Extensions	0	0
31	3DNow! Instructions	1	1

Notes:

1. *Bit definitions: 0 = No Support, 1 = Support.*
2. *The SYSENTER and SYSEXIT instructions have different implementations than the SYSCALL and SYSRET instructions.*

Table 8. Extended Feature Flag Descriptions for the AMD Athlon™ Processors

Bit	Feature[1]	AMD Athlon™ Processor (Model 1)	AMD Athlon™ Processor (Model 2)	AMD Athlon™ Processor (Model 4)
0	Floating-Point Unit	1	1	1
1	Virtual Mode Extensions	1	1	1
2	Debugging Extensions	1	1	1
3	Page Size Extensions (4-Mbyte Pages)	1	1	1
4	Time Stamp Counter (with RDTSC and CR4 disable bit)	1	1	1
5	K86™ Family of Processors' Model-Specific Registers (with RDMSR and WRMSR)	1	1	1
6	PAE (Page Address Extensions)	1	1	1
7	Machine Check Exception	1	1	1
8	CMPXCHG8B Instruction	1	1	1
9	APIC	0	1[2]	1[2]
10	*Reserved on all AMD processors*	0	0	0
11	SYSCALL and SYSRET Instructions[3]	1	1	1
12	Memory Type Range Registers	1	1	1
13	Global Paging Extension	1	1	1
14	Machine Check Architecture	1	1	1
15	Conditional Move Instruction	1	1	1
16	PAT (Page Attribute Table)	1	1	1
17	PSE-36 (Page Size Extensions)	0	1	1
18–21	*Reserved on all AMD processors*	0	0	0
22	AMD MMX™ Instruction Extensions	1	1	1
23	MMX Instructions	1	1	1
24	FXSAVE/FXRSTOR	0	1	1
25–29	*Reserved on all AMD processors*	0	0	0
30	AMD 3DNow!™ Instruction Extensions	1	1	1
31	3DNow! Instructions	1	1	1

Notes:

1. *Bit definitions: 0 = No Support, 1 = Support.*
2. *The AMD Athlon processor Model 2 and Model 4 contain a local APIC. The BIOS must enable the local APIC for bit 9 to return a 1 (supported).*
3. *The SYSENTER and SYSEXIT instructions have different implementations than the SYSCALL and SYSRET instructions.*

Functions 8000_0002h, 8000_0003h, and 8000_0004h — Processor Name String

Input: EAX = 8000_0002h, 8000_0003h, or 8000_0004h

Output: EAX = Processor Name String
EBX = Processor Name String
ECX = Processor Name String
EDX = Processor Name String

Functions 8000_0002h, 8000_0003h, and 8000_0004h each return part of the processor name string in the EAX, EBX, ECX, and EDX registers. These three functions use the four registers to return an ASCII string of up to 48 characters in little endian format. For example, function 8000_0002h returns the first 16 characters of the processor name. The first character resides in the least significant byte of EAX, and the last character (of this group of 16) resides in the most significant byte of EDX. The NULL character (ASCII 00h) is used to indicate the end of the processor name string. This feature is useful for processor names that require fewer than 48 characters.

Function 8000_0005h — L1 TLB/Cache Information for the AMD Athlon™ Processor

Input: EAX = 8000_0005h

Output: EAX = 2-Mbyte/4-Mbyte Pages and L1 TLB Information
EBX = 4-Kbyte Pages and L1 TLB Information
ECX = L1 Data Cache Information
EDX = L1 Instruction Cache Information

Function 8000_0005h returns information about the processor L1 TLBs and caches. Tables 9, 10, 11 and 12 provide the format for the information returned by the 8000_0005h function for the AMD Athlon processor.

Table 9 describes the format of the information for the L1 2-Mbyte/4-Mbyte large page TLBs.

Table 9. EAX Format Returned by Function 8000_0005h

	2-Mbyte/4-Mbyte Pages			
	Data TLB		Instruction TLB	
	Associativity[1]	# Entries[2]	Associativity[1]	# Entries[2]
EAX	Bits 31–24	Bits 23–16	Bits 15–8	Bits 7–0

Notes:

1. *See "Associativity for L1 Caches and L1 TLBs" on page 40 for more information.*
2. *The number of entries returned is the number of entries available for 2-Mbyte large pages. Because 4-Mbyte large pages require two 2-Mbyte entries, the number of entries available for 4-Mbyte large pages is one-half the returned value.*

Table 10. EBX Format Returned by Function 8000_0005h

	4-Kbyte Pages			
	Data TLB		Instruction TLB	
	Associativity*	# Entries	Associativity*	# Entries
EBX	Bits 31–24	Bits 23–16	Bits 15–8	Bits 7–0

Note:

* *See "Associativity for L1 Caches and L1 TLBs" on page 40 for more information.*

Table 11. ECX Format Returned by Function 8000_0005h

	L1 Data Cache			
	Size (Kbytes)	Associativity*	Lines per Tag	Line Size (bytes)
ECX	Bits 31–24	Bits 23–16	Bits 15–8	Bits 7–0

Note:

* *See "Associativity for L1 Caches and L1 TLBs" on page 40 for more information.*

Table 12. EDX Format Returned by Function 8000_0005h

	L1 Instruction Cache			
	Size (Kbytes)	Associativity*	Lines per Tag	Line Size (bytes)
EDX	Bits 31–24	Bits 23–16	Bits 15–8	Bits 7–0

Note:

* *See "Associativity for L1 Caches and L1 TLBs" on page 40 for more information.*

Function 8000_0005h — L1 Cache Information for AMD-K5™ and All AMD-K6® Processors

Input: EAX = 8000_0005h

Output: EAX = Reserved
EBX = TLB Information
ECX = L1 Data Cache Information
EDX = L1 Instruction Cache Information

Function 8000_0005h returns information about the processor's on-chip L1 caches and associated TLBs. Tables 13, 14, and 15 provide the format for the information returned by the 8000_0005h function for the AMD-K5 and all AMD-K6 processors.

Table 13. EBX Format Returned by Function 8000_0005h

	Data TLB		Instruction TLB	
	Associativity*	# Entries	Associativity*	# Entries
EBX	Bits 31–24	Bits 23–16	Bits 15–8	Bits 7–0

Note:
* *See "Associativity for L1 Caches and L1 TLBs" on page 40 for more information.*

Table 14. ECX Format Returned by Function 8000_0005h

	L1 Data Cache			
	Size (Kbytes)	Associativity*	Lines per Tag	Line Size (bytes)
ECX	Bits 31–24	Bits 23–16	Bits 15–8	Bits 7–0

Note:
* *See "Associativity for L1 Caches and L1 TLBs" on page 40 for more information.*

Table 15. EDX Format Returned by Function 8000_0005h

	L1 Instruction Cache			
	Size (Kbytes)	Associativity*	Lines per Tag	Line Size (bytes)
EDX	Bits 31–24	Bits 23–16	Bits 15–8	Bits 7–0

Note:
* *See "Associativity for L1 Caches and L1 TLBs" on page 40 for more information.*

Function 8000_0006h — L2 TLB/L2 Cache Information for the AMD Athlon™ Processor

Note: *The AMD Athlon processor L2 cache must be configured prior to invoking this function.*

Input: EAX = 8000_0006h

Output: EAX = 2-Mbyte/4-Mbyte Pages and L2 TLB Information
EBX = 4-Kbyte Pages and L2 TLB Information
ECX = L2 Unified Cache Information
EDX = Reserved

Function 8000_0006h returns information about the L2 cache and TLB. Tables 16, 17, and 18 provide the format for the information returned by the 8000_0006h function on the AMD Athlon processor.

Table 16. EAX Format Returned by Function 8000_0006h

	2-Mbyte/4-Mbyte Pages			
	L2 Data TLB[2]		L2 Instruction or Unified TLB	
	Associativity[1]	# Entries	Associativity[1]	# Entries
EAX	Bits 31–28	Bits 27–16	Bits 15–12	Bits 11–0

Notes:
1. *See "Associativity for L2 Caches and L2 TLBs" on page 40 for more information.*
2. *A unified L2 TLB is indicated by a value of 0000h in the upper 16 bits of the EBX register. Unified TLB information is then referenced in the lower 16 bits of the EBX register.*

Table 17. EBX Format Returned by Function 8000_0006h

	4-Kbyte Pages			
	L2 Data TLB[2]		L2 Instruction or Unified TLB	
	Associativity[1]	# Entries	Associativity[1]	# Entries
EBX	Bits 31–28	Bits 27–16	Bits 15–12	Bits 11–0

Notes:
1. *See "Associativity for L2 Caches and L2 TLBs" on page 40 for more information.*
2. *A unified L2 TLB is indicated by a value of 0000h in the upper 16 bits of the EBX register. Unified TLB information is then referenced in the lower 16 bits of the EBX register.*

Table 18. ECX Format Returned by Function 8000_0006h

	L2 Cache			
	Size (Kbytes)	Associativity*	Lines per Tag	Line Size (bytes)
ECX	Bits 31–16	Bits 15–12	Bits 11–8	Bits 7–0

Note:
* *See "Associativity for L2 Caches and L2 TLBs" on page 40 for more information.*

Function 8000_0006h – L2 Cache Information for the AMD-K6®-III Processor

Input: EAX = 8000_0006h

Output: EAX = Reserved
EBX = Reserved
ECX = L2 Unified Cache Information
EDX = Reserved

Function 8000_0006h returns information about the processor's L2 cache. Table 19 provides the format for the information returned by the 8000_0006h function.

Table 19. ECX Format Returned by Function 8000_0006h for the AMD-K6®-III Processor

	L2 Cache			
	Size (Kbytes)	Associativity*	Lines per Tag	Line Size (bytes)
ECX	Bits 31–16	Bits 15–12	Bits 11–8	Bits 7–0

Note:
* *See "Associativity for L2 Caches and L2 TLBs" on page 40 for more information.*

Associativity Field Definitions

This section describes the values returned in the associativity fields.

Associativity for L1 Caches and L1 TLBs

The associativity fields for the L1 data cache, L1 instruction cache, L1 data TLB, and L1 instruction TLB are all 8 bits wide. Except for 00h (Reserved) and FFh (Full), the number returned in the associativity field represents the actual number of ways, with a range of 01h through FEh. For example, a returned value of 02h indicates 2-way associativity and a returned value of 04h indicates 4-way associativity.

Associativity for L2 Caches and L2 TLBs

The associativity fields for the L2 cache, L2 data TLB, and L2 instruction TLB are 4 bits wide. Table 20 shows the values returned in these associativity fields.

Table 20. Associativity Values For L2 Caches and TLBs

Bits 15–12	Associativity
0000b	L2 off
0001b	Direct mapped
0010b	2-way
0011b	Reserved
0100b	4-way
0101b	Reserved
0110b	8-way
0111b	Reserved
1000b	16-way
1001b	Reserved
1010b	Reserved
1011b	Reserved
1100b	Reserved
1101b	Reserved
1110b	Reserved
1111b	Full

Appendix B

Register Values Returned by the AMD Family Processors

Tables 21 through 23 contain all the values returned for AMD processors by the CPUID instruction.

AMD Athlon™ Processor Values

Table 21. Values Returned By the AMD Athlon™ Processor

Function / Register	AMD Athlon™ Processor (Model 1)	AMD Athlon™ Processor (Model 2)	AMD Athlon™ Processor (Model 4)
Function: 0			
EAX	0000_0001h	0000_0001h	0000_0001h
EBX	6874_7541h	6874_7541h	6874_7541h
ECX	444D_4163h	444D_4163h	444D_4163h
EDX	6974_6E65h	6974_6E65h	6974_6E65h
Function: 1			
EAX	0000_061Xh	0000_062Xh	0000_064Xh
EBX	Reserved	Reserved	Reserved
ECX	Reserved	Reserved	Reserved
EDX	0081_F9FFh	0183_F9FFh[1]	0183_F9FFh[1]
Function: 8000_0000h			
EAX	8000_0006h	8000_0006h	8000_0006h
EBX	Reserved	Reserved	Reserved
ECX	Reserved	Reserved	Reserved
EDX	Reserved	Reserved	Reserved
Function: 8000_0001h			
EAX	0000_071Xh	0000_072Xh	0000_074Xh
EBX	Reserved	Reserved	Reserved
ECX	Reserved	Reserved	Reserved
EDX	C0C1_F9FFh	C1C3_F9FFh[2]	C1D3_F9FFh[2]
Function: 8000_0002h			
EAX	2D44_4D41h	2044_4D41h	2044_4D41h
EBX	7428_374Bh	6C68_7441h	6C68_7441h
ECX	5020_296Dh	7428_6E6Fh	7428_6E6Fh
EDX	6563_6F72h	5020_296Dh	5020_296Dh

Notes:

1. *The AMD Athlon processor (Models 2 and 4) contains an APIC, but it is not reflected in this value. If the BIOS enables the APIC, the value is 0183_FBFFh.*
2. *The AMD Athlon processor (Models 2 and 4) contains an APIC, but it is not reflected in this value. If the BIOS enables the APIC, the value is C1C3_FBFFh.*
3. *The L2 cache size and associativity on AMD processors are product specific. The AMD Athlon processor (Models 1 and 2) have an L2 cache size of 512-Kbyte with two-way set associativity. The AMD Athlon processor Model 4 has an L2 cache size of 256-Kbyte with sixteen-way set associativity.*

Table 21. Values Returned By the AMD Athlon™ Processor (continued)

Function / Register	AMD Athlon™ Processor (Model 1)	AMD Athlon™ Processor (Model 2)	AMD Athlon™ Processor (Model 4)
Function: 8000_0003h			
EAX	726F_7373h	6563_6F72h	6563_6F72h
EBX	0000_0000h	726F_7373h	726F_7373h
ECX	0000_0000h	0000_0000h	0000_0000h
EDX	0000_0000h	0000_0000h	0000_0000h
Function: 8000_0004h			
EAX	0000_0000h	0000_0000h	0000_0000h
EBX	0000_0000h	0000_0000h	0000_0000h
ECX	0000_0000h	0000_0000h	0000_0000h
EDX	0000_0000h	0000_0000h	0000_0000h
Function: 8000_0005h			
EAX	0408_FF08h	0408_FF08h	0408_FF08h
EBX	FF18_FF10h	FF18_FF10h	FF18_FF10h
ECX	4002_0140h	4002_0140h	4002_0140h
EDX	4002_0140h	4002_0140h	4002_0140h
Function: 8000_0006h			
EAX	0000_0000h	0000_0000h	0000_0000h
EBX	4100_4100h	4100_4100h	4100_4100h
ECX	****_*140h[3]	****_*140h[3]	****_*140h[3]
EDX	Reserved	Reserved	Reserved

Notes:

1. *The AMD Athlon processor (Models 2 and 4) contains an APIC, but it is not reflected in this value. If the BIOS enables the APIC, the value is 0183_FBFFh.*
2. *The AMD Athlon processor (Models 2 and 4) contains an APIC, but it is not reflected in this value. If the BIOS enables the APIC, the value is C1C3_FBFFh.*
3. *The L2 cache size and associativity on AMD processors are product specific. The AMD Athlon processor (Models 1 and 2) have an L2 cache size of 512-Kbyte with two-way set associativity. The AMD Athlon processor Model 4 has an L2 cache size of 256-Kbyte with sixteen-way set associativity.*

AMD-K6® Processor Values

Table 22. Values Returned By AMD-K6® Processors

Function / Register	AMD-K6® Processor (Model 6)	AMD-K6 Processor (Model 7)	AMD-K6-2 Processor (Model 8)	AMD-K6-III Processor (Model 9)
Function: 0				
EAX	0000_0001h	0000_0001h	0000_0001h	0000_0001h
EBX	6874_7541h	6874_7541h	6874_7541h	6874_7541h
ECX	444D_4163h	444D_4163h	444D_4163h	444D_4163h
EDX	6974_6E65h	6974_6E65h	6974_6E65h	6974_6E65h
Function: 1				
EAX	0000_056Xh	0000_057Xh	0000_058Xh	0000_059Xh
EBX	Reserved	Reserved	Reserved	Reserved
ECX	Reserved	Reserved	Reserved	Reserved
EDX	0080_01BFh	0080_01BFh	0080_21BFh[1]	0080_21BFh
Function: 8000_0000h				
EAX	8000_0005h	8000_0005h	8000_0005h	8000_0006h
EBX	Reserved	Reserved	Reserved	Reserved
ECX	Reserved	Reserved	Reserved	Reserved
EDX	Reserved	Reserved	Reserved	Reserved
Function: 8000_0001h				
EAX	0000_066Xh	0000_067Xh	0000_068Xh	0000_069Xh
EBX	Reserved	Reserved	Reserved	Reserved
ECX	Reserved	Reserved	Reserved	Reserved
EDX	0080_05BFh	0080_05BFh	8080_29BFh[2]	8080_29BFh

Notes:

1. *AMD-K6-2 processor Model 8/[F:8], EDX = 0080_21BFh – Global Paging Extension supported. AMD-K6-2 processor Model 8/[7:0], EDX = 0080_01BFh.*
2. *AMD-K6-2 processor Model 8/[F:8], EDX = 8080_29BFh – Global Paging Extension supported. AMD-K6-2 processor Model 8/[7:0], EDX = 8080_09BFh.*
3. *Extended functions 8000_0002h, 8000_0003h, and 8000_0004h each return part of the processor name string. Some AMD-K6-2 processors may have the following name string: function 8000_0002h, ECX = 322D_296Dh and EDX = 6F72_5020h, and function 8000_0003h, EAX = 7373_6563h and EBX = 0000_726Fh.*
4. *Extended functions 8000_0002h, 8000_0003h, and 8000_0004h each return part of the processor name string. Some AMD-K6-III processors may have the following name string: function 8000_0002h, ECX = 492D_296Dh and EDX = 5020_4949h, and function 8000_0003h, EAX = 6563_6F72h and EBX = 726F_7373h.*

Table 22. Values Returned By AMD-K6® Processors (continued)

Function / Register	AMD-K6® Processor (Model 6)	AMD-K6 Processor (Model 7)	AMD-K6-2 Processor (Model 8)	AMD-K6-III Processor (Model 9)
Function: 8000_0002h				
EAX	2D44_4D41h	2D44_4D41h	2D44_4D41h	2D44_4D41h
EBX	6D74_364Bh	6D74_364Bh	7428_364Bh	7428_364Bh
ECX	202F_7720h	202F_7720h	3320_296Dh[3]	3320_296Dh[4]
EDX	746C_756Dh	746C_756Dh	7270_2044h[3]	5020_2B44h[4]
Function: 8000_0003h				
EAX	6465_6D69h	6465_6D69h	7365_636Fh[3]	6563_6F72h[4]
EBX	6520_6169h	6520_6169h	0072_6F73h[3]	726F_7373h[4]
ECX	6E65_7478h	6E65_7478h	0000_0000h	0000_0000h
EDX	6E6F_6973h	6E6F_6973h	0000_0000h	0000_0000h
Function: 8000_0004h				
EAX	0000_0073h	0000_0073h	0000_0000h	0000_0000h
EBX	0000_0000h	0000_0000h	0000_0000h	0000_0000h
ECX	0000_0000h	0000_0000h	0000_0000h	0000_0000h
EDX	0000_0000h	0000_0000h	0000_0000h	0000_0000h
Function: 8000_0005h				
EAX	Reserved	Reserved	Reserved	Reserved
EBX	0280_0140h	0280_0140h	0280_0140h	0280_0140h
ECX	2002_0220h	2002_0220h	2002_0220h	2002_0220h
EDX	2002_0220h	2002_0220h	2002_0220h	2002_0220h

Notes:

1. *AMD-K6-2 processor Model 8/[F:8], EDX = 0080_21BFh – Global Paging Extension supported. AMD-K6-2 processor Model 8/[7:0], EDX = 0080_01BFh.*
2. *AMD-K6-2 processor Model 8/[F:8], EDX = 8080_29BFh – Global Paging Extension supported. AMD-K6-2 processor Model 8/[7:0], EDX = 8080_09BFh.*
3. *Extended functions 8000_0002h, 8000_0003h, and 8000_0004h each return part of the processor name string. Some AMD-K6-2 processors may have the following name string: function 8000_0002h, ECX = 322D_296Dh and EDX = 6F72_5020h, and function 8000_0003h, EAX = 7373_6563h and EBX = 0000_726Fh.*
4. *Extended functions 8000_0002h, 8000_0003h, and 8000_0004h each return part of the processor name string. Some AMD-K6-III processors may have the following name string: function 8000_0002h, ECX = 492D_296Dh and EDX = 5020_4949h, and function 8000_0003h, EAX = 6563_6F72h and EBX = 726F_7373h.*

Table 22. Values Returned By AMD-K6® Processors (continued)

Function / Register	AMD-K6® Processor (Model 6)	AMD-K6 Processor (Model 7)	AMD-K6-2 Processor (Model 8)	AMD-K6-III Processor (Model 9)
Function: 8000_0006h				
EAX	Undefined	Undefined	Undefined	Reserved
EBX	Undefined	Undefined	Undefined	Reserved
ECX	Undefined	Undefined	Undefined	0100_4220h
EDX	Undefined	Undefined	Undefined	Reserved

Notes:

1. *AMD-K6-2 processor Model 8/[F:8], EDX = 0080_21BFh – Global Paging Extension supported. AMD-K6-2 processor Model 8/[7:0], EDX = 0080_01BFh.*
2. *AMD-K6-2 processor Model 8/[F:8], EDX = 8080_29BFh – Global Paging Extension supported. AMD-K6-2 processor Model 8/[7:0], EDX = 8080_09BFh.*
3. *Extended functions 8000_0002h, 8000_0003h, and 8000_0004h each return part of the processor name string. Some AMD-K6-2 processors may have the following name string: function 8000_0002h, ECX = 322D_296Dh and EDX = 6F72_5020h, and function 8000_0003h, EAX = 7373_6563h and EBX = 0000_726Fh.*
4. *Extended functions 8000_0002h, 8000_0003h, and 8000_0004h each return part of the processor name string. Some AMD-K6-III processors may have the following name string: function 8000_0002h, ECX = 492D_296Dh and EDX = 5020_4949h, and function 8000_0003h, EAX = 6563_6F72h and EBX = 726F_7373h.*

Am486®, Am5$_X$86®, and AMD-K5™ Processor Values

Table 23. Values Returned By Am486®, Am5$_X$86®, and AMD-K5™ Processors

Function Register	Am486® and Am5$_X$86® Processors	AMD-K5™ Processor (Model 0)	AMD-K5 Processor (Model 1)	AMD-K5 Processor (Model 2)	AMD-K5 Processor (Model 3)
Function: 0					
EAX	0000_0001h	0000_0001h	0000_0001h	0000_0001h	0000_0001h
EBX	6874_7541h	6874_7541h	6874_7541h	6874_7541h	6874_7541h
ECX	444D_4163h	444D_4163h	444D_4163h	444D_4163h	444D_4163h
EDX	6974_6E65h	6974_6E65h	6974_6E65h	6974_6E65h	6974_6E65h
Function: 1					
EAX	0000_04XXh	0000_050Xh	0000_051Xh	0000_052Xh	0000_053Xh
EBX	Reserved	Reserved	Reserved	Reserved	Reserved
ECX	Reserved	Reserved	Reserved	Reserved	Reserved
EDX	0000_0001h	0000_03BFh*	0000_21BFh	0000_21BFh	0000_21BFh
Function: 8000_0000h					
EAX	0000_0000h	0000_0000h	8000_0005h	8000_0005h	8000_0005h
EBX	Undefined	Undefined	Reserved	Reserved	Reserved
ECX	Undefined	Undefined	Reserved	Reserved	Reserved
EDX	Undefined	Undefined	Reserved	Reserved	Reserved
Function: 8000_0001h					
EAX	Undefined	Undefined	0000_051Xh	0000_052Xh	0000_053Xh
EBX	Undefined	Undefined	Reserved	Reserved	Reserved
ECX	Undefined	Undefined	Reserved	Reserved	Reserved
EDX	Undefined	Undefined	0000_21BFh	0000_21BFh	0000_21BFh
Function: 8000_0002h					
EAX	Undefined	Undefined	2D44_4D41h	2D44_4D41h	2D44_4D41h
EBX	Undefined	Undefined	7428_354Bh	7428_354Bh	7428_354Bh
ECX	Undefined	Undefined	5020_296Dh	5020_296Dh	5020_296Dh
EDX	Undefined	Undefined	6563_6F72h	6563_6F72h	6563_6F72h
Function: 8000_0003h					
EAX	Undefined	Undefined	726F_7373h	726F_7373h	726F_7373h
EBX	Undefined	Undefined	0000_0000h	0000_0000h	0000_0000h
ECX	Undefined	Undefined	0000_0000h	0000_0000h	0000_0000h
EDX	Undefined	Undefined	0000_0000h	0000_0000h	0000_0000h

Note:

* *The AMD-K5 processor Model 0 reserves bit 13 and implements feature bit 9 to indicate support for Global Paging Extensions instead of support for APIC.*

Table 23. Values Returned By Am486®, Am5$_X$86®, and AMD-K5™ Processors (continued)

Function Register	Am486® and Am5$_X$86® Processors	AMD-K5™ Processor (Model 0)	AMD-K5 Processor (Model 1)	AMD-K5 Processor (Model 2)	AMD-K5 Processor (Model 3)
Function: 8000_0004h					
EAX	Undefined	Undefined	0000_0000h	0000_0000h	0000_0000h
EBX	Undefined	Undefined	0000_0000h	0000_0000h	0000_0000h
ECX	Undefined	Undefined	0000_0000h	0000_0000h	0000_0000h
EDX	Undefined	Undefined	0000_0000h	0000_0000h	0000_0000h
Function: 8000_0005h					
EAX	Undefined	Undefined	Reserved	Reserved	Reserved
EBX	Undefined	Undefined	0480_0000h	0480_0000h	0480_0000h
ECX	Undefined	Undefined	0804_0120h	0804_0120h	0804_0120h
EDX	Undefined	Undefined	1004_0120h	1004_0120h	1004_0120h

Note:

* *The AMD-K5 processor Model 0 reserves bit 13 and implements feature bit 9 to indicate support for Global Paging Extensions instead of support for APIC.*