インテル® プロセッサの識別とCPUID命令

アプリケーション・ノート 485

2005 年 3 月

本資料に掲載されている情報は、インテル製品の概要説明を目的としたものです。本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスを許諾するためのものではありません。製品に付属の売買契約書『Intel's Terms and conditions of Sales』に規定されている場合を除き、インテルはいかなる責を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証（特定目的への適合性、商品性に関する保証、第三者の特許権、著作権、その他、知的所有権を侵害していないことへの保証を含む）に関しても一切責任を負わないものとします。インテル製品は、医療、救命、延命措置などの目的への使用を前提としたものではありません。

インテル製品は、予告なく仕様が変更される場合があります。

機能または命令の一覧で「予約」または「未定義」と記されているものがありますが、その「機能が存在しない」という状態を設計の前提にするのはおやめください。これらの項目は、インテルが将来のために予約しているものです。インテルが将来これらの項目を定義したことにより、衝突が生じたり互換性が失われたりしても、インテルは一切責任を負わないものとします。

インテル® プロセッサは、エラッタと呼ばれる設計上の不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。現在確認済みのエラッタについては、インテルまでお問い合わせください。

最新の仕様をご希望の場合や製品をご注文の場合は、お近くのインテルの営業所または販売代理店にお問い合わせください。

† ハイパー・スレッディング・テクノロジを利用するには、ハイパー・スレッディング・テクノロジに対応したインテル® Pentium® 4 プロセッサを搭載したコンピュータ・システム、および同技術に対応したチップセットと BIOS、OS が必要です。性能は、使用するハードウェアやソフトウェアによって異なります。HT テクノロジに対応したプロセッサの情報等、詳細については http://www.intel.co.jp/jp/info/hyperthreading/ を参照してください。

目 次

図

表

例

改訂履歴

改訂番号	説明	改訂時期
-001	初版	1993 年 5 月
-002	表 4「Intel486™ プロセッサと Pentium® プロセッサのシグニチャ」を変更。	1993 年 10 月
-003	プロセッサの新しいバージョンに合わせて改訂。プログラムの例を使いやすいように変更。OverDrive® プロセッサの BIOS 認識に関する項を追加。機能フラグの情報を改訂。	1994 年 9 月
-004	Pentium® Pro プロセッサと OverDrive プロセッサの情報を改訂。表 1、表 3、表 5 を変更。表 6、表 9、表 10 を挿入。3.4. 節と 3.5. 節を挿入。	1995 年 12 月
-005	図 1 と図 3 を追加。脚注 1 と脚注 2 を追加。図 2 を変更。第 4 章にアセンブリ・コードの例を追加。表 3、表 5、表 7 を追加。5.0 節に 2 つの項目を追加。プロセッサが MMX® テクノロジに対応しているかを判定するように、cpuid3b.ASM と cpuid3b.C のプログラムを変更。図 6.0 を変更。	1996 年 11 月
-006	表 3 を変更。将来の P6 ファミリ・プロセッサに備えて予約済みのエントリを追加。Pentium® II プロセッサ・ファミリを示すように表の見出しを変更。表 5 を変更。SEP ビットの定義を追加。3.5. 節を追加。3.7 節と表 9 を追加。正しい使用法を示すように P6 ファミリのリファレンスを修正。 SEP 機能ビットのチェックと Pentium II プロセッサのチェック / 識別を行うように、cpuid3a.asm、cpuid3b.asm、cpuid3.c のコード例を変更。設計者とエラッタに関する免責条項を追加。	1997 年 3 月
-007	表 2 を変更。Pentium II プロセッサ、モデル 5 のエントリを追加。既存の Pentium II プロセッサのエントリを変更し、「Pentium II プロセッサ、モデル 3」を表示。表 5 を変更。追加の機能ビット PAT と FXSR を追加。表 7 を変更。エントリ 44h と 45h を追加。 4.0 節の注「機能フラグの値が 1 の場合は特定の機能が常に存在するとは考えないでください。将来の機能フラグでは、1 の値は特定の機能が存在しないことを示す可能性があります。」を削除。 Pentium II プロセッサ、モデル 5 のチェックと識別を行うように、cpuid3b.asm と cpuid3.c のコード例を変更。Pentium II プロセッサ、モデル 3 を出力するように、既存の Pentium II プロセッサ・コードを変更。	1998 年 1 月
-008	3.2 節にインテル® Celeron® プロセッサ、モデル 5 の識別に関する注を追加。表 2 を変更。Celeron プロセッサと MMX® テクノロジ Pentium® OverDrive® プロセッサのエントリを追加。表 5 を変更。追加の機能ビット PSE-36 を追加。 Celeron プロセッサのチェックと識別を行うように、cpuid3b.asm と cpuid3.c のコード例を変更。	1998 年 4 月
-009	3.2 節に Pentium® II Xeon™ プロセッサの識別に関する注を追加。表 2 を変更。Pentium II Xeon プロセッサのエントリを追加。 Pentium II Xeon プロセッサのチェックと識別を行うように、cpuid3b.asm と cpuid3.c のコード例を変更。	1998 年 6 月
-010	変更なし	

改訂番号	説明	改訂時期
-011	表 2 を変更。Celeron プロセッサ、モデル 6 のエントリを追加。 Celeron プロセッサ、モデル 6 のチェックと識別を行うように、cpuid3b.asm と cpuid3.c のコード例を変更。	1998 年 12 月
-012	図 1 を変更し、Intel386™ プロセッサの予約済み情報を追加。図 2 を変更。EAX=3 に設定して CPUID 命令を実行した場合に返されるプロセッサ・シリアル・ナンバ情報を追加。表 1 を変更。プロセッサ・シリアル・ナンバ・パラメータを追加。表 2 を変更。Pentium® III プロセッサと Pentium® III Xeon™ プロセッサを追加。第 4 章「プロセッサ・シリアル・ナンバ」を追加。 Pentium III プロセッサと Pentium III Xeon プロセッサのチェックと識別を行うように、cpuid3a.asm、cpuid3b.asm、cpuid3.c のコード例を変更。	1998 年 12 月
-013	図 2 を変更。EAX=1 に設定して CPUID 命令を実行した場合に返されるブランド ID 情報を追加。第 5 章「ブランド ID」を追加。表 10 を追加し、定義済みのブランド ID 値を表示。 Pentium III プロセッサ、モデル 8 と Pentium III Xeon プロセッサ、モデル 8 のチェックと識別を行うように、cpuid3a.asm、cpuid3b.asm、cpuid3.c のコード例を変更。	1999 年 10 月
-014	表 4 を変更。Celeron プロセッサ、モデル 8 を追加。	2000 年 3 月
-015	表 4 を変更。Pentium III Xeon プロセッサ、モデル A を追加。表 10 を変更。8 ウェイ・セット・アソシアティブ 1M キャッシュ・ディスクリプタと 8 ウェイ・セット・アソシアティブ 2M キャッシュ・ディスクリプタのエントリを追加。	2000 年 5 月
-016	図 2 を改訂し、EAX=1 に設定して CPUID を実行した場合の拡張ファミリと拡張モデルを追加。 第 6 章を追加し、ブランド・ストリングについて説明。 第 10 章「機能検出の代替手法」とコード例を追加。 表 4 に Pentium® 4 プロセッサ・シグニチャを追加。 表 5 に新しい機能フラグ（SSE2、SS、TM）を追加。 表 10 に新しいキャッシュ・ディスクリプタを追加。 Pentium Pro プロセッサのキャッシュ・ディスクリプタの例を削除。	2000 年 11 月
-017	図 2 を変更し、Pentium 4 プロセッサによって報告される追加機能を追加。 表 10 を変更し、Intel NetBurst® マイクロアーキテクチャによって定義される追加のキャッシュ・ディスクリプタと TLB ディスクリプタを追加。 第 10 章とプログラム例 5 を追加し、プロセッサの DAZ 機能サポートの検出方法について説明。 第 11 章とプログラム例 6 を追加し、プロセッサの実際の動作周波数の計算方法について説明。	2001 年 2 月
-018	表 7 の 2 番目の 66h キャッシュ・ディスクリプタを 68h に変更。 表 7 に 83h キャッシュ・ディスクリプタを追加。 表 4 に Pentium III プロセッサ、モデル B のプロセッサ・シグニチャとインテル Xeon プロセッサのプロセッサ・シグニチャを追加。 表 4 を変更し、拡張ファミリと拡張モデルのフィールドを追加。 表 1 を変更し、拡張された CPUID 機能によって返される情報を追加。	2001 年 6 月

改訂番号	説明	改訂時期
-019	資料全体を通してインテル Celeron 商標を使用するように変更。 表 12 を変更し、Intel NetBurst マイクロアーキテクチャを採用したインテル・プロセッサによってサポートされる新しいブランド ID の値を追加。 表 5 にハイパー・スレッディング・テクノロジ・フラグを追加。表 1 に論理プロセッサ数を追加。 表 12 の改訂されたブランド ID の値に基づいてインテル・プロセッサのチェックと識別を行うように、cpuid3b.asm と cpuid3.c のコード例を変更。	2002 年 1 月
-020	表 10 を変更し、Intel NetBurst マイクロアーキテクチャを採用したインテル・プロセッサによってサポートされる新しいキャッシュ・ディスクリプタの値を追加。 表 12 の改訂されたブランド ID の値に基づいてインテル・プロセッサのチェックと識別を行うように、cpuid3b.asm と cpuid3.c のコード例を変更。	2002 年 3 月
-021	表 4 を変更し、ファミリコードの値が 0Fh になるプロセッサ・シグニチャを返すプロセッサを追加。 表 10 を変更し、各種のインテル・プロセッサによってサポートされる新しいキャッシュ・ディスクリプタの値を追加。 表 12 を変更し、Intel NetBurst マイクロアーキテクチャを採用したインテル・プロセッサによってサポートされる新しいブランド ID の値を追加。 表 12 の改訂されたブランド ID の値に基づいてインテル・プロセッサのチェックと識別を行うように、cpuid3b.asm と cpuid3.c のコード例を変更。	2002 年 5 月
-022	表 10 を変更し、正しいキャッシュ・ディスクリプタの説明を追加。 表 5 を変更し、EDX に返される新しい機能フラグを追加。 表 6 に ECX に返される機能フラグを追加。 表 4 を変更し、ファミリ「F」のプロセッサをモデル番号ごとに分類。	2002 年 11 月
-023	表 4 を変更し、インテル® Pentium® M プロセッサを追加。 表 5 を変更し、EDX に返される新しい機能フラグを追加。 表 6 を変更し、ECX に返される機能フラグを追加。 表 10 を変更し、正しいキャッシュ・ディスクリプタの説明を追加。	2003 年 3 月
-024	表 6 のビット 7 とビット 8 の機能フラグの定義を修正。	2003 年 11 月

改訂番号	説明	改訂時期
-025	表 1 を変更し、決定性キャッシュ・パラメータ機能（EAX=4 で CPUID 命令を実行）、MONITOR/MWAIT 機能（EAX=5 で CPUID 命令を実行）、拡張 L2 キャッシュ機能（EAX=80000006 で CPUID 命令を実行）、拡張アドレスサイズ機能（EAX=80000008 で CPUID 命令を実行）を追加。 表 1 と表 5 を変更し、Pentium 4 プロセッサ・ファミリには PSN がないことを強調。 90nm プロセスを採用したインテル Pentium 4 プロセッサとインテル Celeron プロセッサを追加。 表 6 を変更し、ECX に返される新しい機能フラグを追加。 表 10 を変更し、各種のインテル・プロセッサによってサポートされる新しいキャッシュ・ディスクリプタの値を追加。 表 12 を変更し、Intel NetBurst マイクロアーキテクチャを採用したインテル・プロセッサによってサポートされる新しいブランド ID の値を追加。 表 12 の改訂されたブランド ID の値に基づいてインテル・プロセッサのチェックと識別を行うように、cpuid3b.asm と cpuid3.c のコード例を変更。 表 6 の改訂された値に基づいて新しい機能フラグのチェックと識別を行うように、features.cpp、cpuid3.c、cpuid3a.asm を変更。	2004 年 1 月
-026	EAX の値を 1 に設定して CPUID 命令を実行した場合に EDX[31]（PBE）に返される機能フラグ名を修正。 表 1 を変更し、CPUID 機能 80000001h によって EAX レジスタに拡張機能フラグが返されることを表示。 表 4 にインテル Pentium M プロセッサ（ファミリ 6、モデル D）を追加。 3.4 節「拡張機能フラグ」と表 7 を追加。 表 6 を変更し、ECX に返される新しい機能フラグを追加。 表 10 を変更し、各種のインテル・プロセッサによってサポートされる新しいキャッシュ・ディスクリプタの値を追加。 表 12 を変更し、P6 ファミリ・マイクロアーキテクチャを採用したインテル・プロセッサによってサポートされる新しいブランド ID の値を追加。 表 12 の改訂されたブランド ID の値に基づいてインテル・プロセッサのチェックと識別を行うように、cpuid3b.asm と cpuid3.c のコード例を変更。 表 6 の改訂された値に基づいて新しい機能フラグのチェックと識別を行うように、features.cpp、cpuid3.c、cpuid3a.asm を変更。	2004 年 5 月
-027	表 7 の拡張機能フラグに使用されるレジスタを修正。	2004 年 7 月

改訂番号	説明	改訂時期
-028	表 1 の CPUID 機能 80000001h と 80000006h のビット・フィールドの定義を修正。 表 4 にファミリ「F」、モデル「4」のプロセッサ名を追加。 表 6 を改訂し、CMPXCHG16B 命令の機能フラグの定義（ECX[13]）を追加。 表 1 を改訂し、SYSCALL / SYSRET（EDX[11]）とエグゼキュート・ディスエーブル・ビット（EDX[20]）の拡張機能フラグの定義を追加。 CPUID 拡張機能情報を抽出するように、表 1 を改訂。 CPUID 機能 80000001h によって識別される拡張機能を検出し、表示するように、例 2 と例 3 を改訂。	2005 年 2 月
-029	表 10 を変更し、各種のインテル・プロセッサによってサポートされる新しいキャッシュ・ディスクリプタの値を追加。	2005 年 3 月

1 はじめに

インテル® アーキテクチャが新しい世代 / モデルのプロセッサの開発とともに進化するにつれて（8086、8088、Intel286、Intel386™ プロセッサ、Intel486™ プロセッサ、Pentium® プロセッサ、Pentium® OverDrive® プロセッサ、MMX® テクノロジ Pentium® プロセッサ、MMX® テクノロジ Pentium® OverDrive® プロセッサ、Pentium® Pro プロセッサ、Pentium® II プロセッサ、Pentium® II Xeon™ プロセッサ、Pentium® II Overdrive® プロセッサ、インテル® Celeron® プロセッサ、モバイル インテル® Celeron® プロセッサ、Pentium® III プロセッサ、モバイル インテル® Pentium® III プロセッサ - M、Pentium® III Xeon™ プロセッサ、Pentium® 4 プロセッサ、モバイル インテル® Pentium® 4 プロセッサ - M、インテル® Pentium® M プロセッサ、インテル® Xeon™ プロセッサ、インテル® Xeon™ プロセッサ MP）、ソフトウェアが各プロセッサ上で利用可能な機能を識別できるように、インテルはますます高度な手段の提供を求められる。こうした識別機構は、インテル・アーキテクチャとともに次のように進化してきた。

1. 当初、インテルは、実装上またはアーキテクチャ上の小さな違いを検出することによってプロセッサの世代を識別するコード・シーケンスを公開していた。
2. その後、Intel386 プロセッサの出現とともに、インテルは、プロセッサのファミリ、モデル、ステッピング番号を（リセット時にのみ）ソフトウェアに提供する、プロセッサ・シグニチャ識別機能を導入した。
3. インテル・アーキテクチャの進化とともに、インテルは、プロセッサ・シグニチャ識別機能を CPUID 命令として拡張した。CPUID 命令は、プロセッサ・シグニチャを提供するだけでなく、インテル・プロセッサがサポートおよび実装している機能に関する情報も提供する。

インテル・アーキテクチャの種類が増えるにつれて、コンピューティング市場では、異なる機能セットを備えたプロセッサ世代 / モデル間でプロセッサの機能を調整する必要が生じたため、プロセッサ識別機能の進化が必要になった。将来の世代のプロセッサでもこの傾向が続くことを予想して、インテル・アーキテクチャは、拡張可能な機能として CPUID 命令を実装している。

本書では、ソフトウェア・アプリケーション、各種の BIOS、各種のプロセッサ・ツール内での CPUID 命令の使用方法について説明する。ソフトウェア開発者は、CPUID 命令の利用により、過去、現在、未来にわたる多様なインテル・プロセッサの世代とモデルで互換的に動作するソフトウェア・アプリケーションやツールを制作できる。

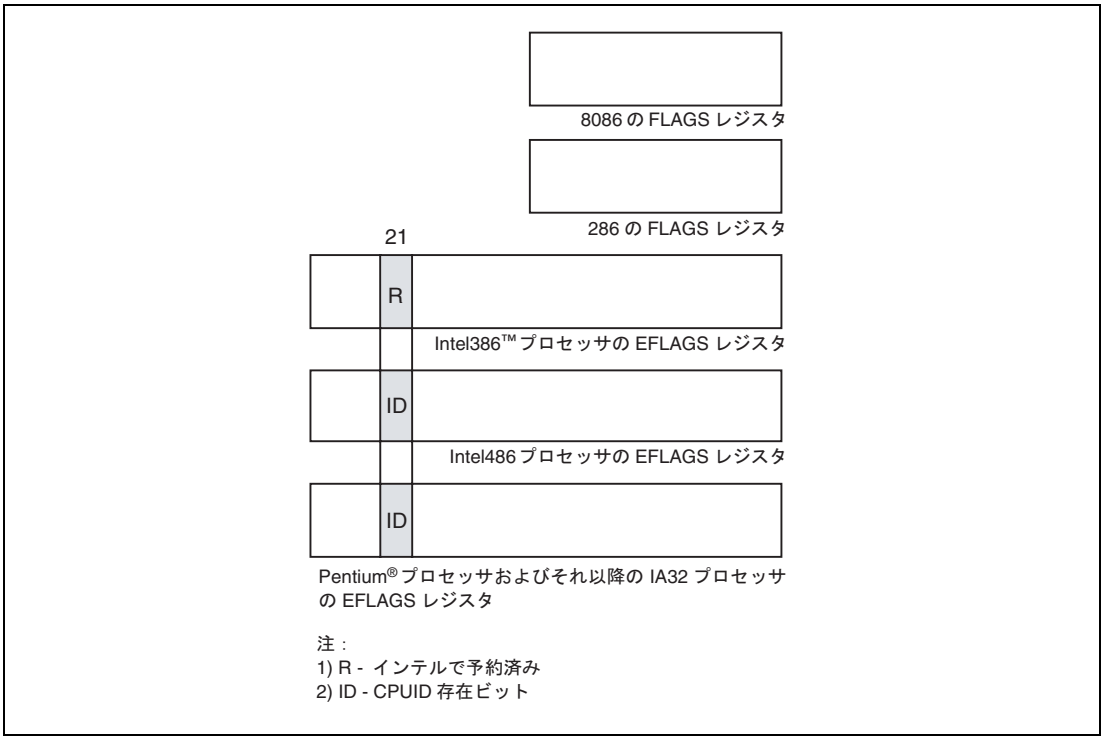
1.1 最新情報の提供

インテル・プロセッサのシグニチャと機能ビットに関する新しい情報は、デベロッパーズ・マニュアル、プログラマーズ・リファレンス・マニュアル、または当該プロセッサの適切なマニュアルから得ることができる。また、本書に記載されているプログラミング例の改訂版も提供している。詳細については、御社担当のインテル社員に問い合わせるか、インテルの Web サイト http://www.intel.co.jp/jp/developer/ を参照のこと。

2 CPUID 命令の検出

Intel486™ ファミリおよびそれ以降のインテル® プロセッサは、プロセッサの内部アーキテクチャが CPUID 命令を実行できるかを判定するための直接的な方法を用意している。この方法は、EFLAGS レジスタのビット 21 の ID フラグを使用する。ソフトウェアがこのフラグの値を変更できれば、CPUID 命令が実行可能である[1]（図 1 を参照）。

図 1. フラグレジスタの進化



POPF、POPFD、PUSHF、PUSHFD 命令を使用して、EFLAGS レジスタ内のフラグにアクセスできる。本書の巻末のプログラム例は、PUSHFD 命令を使用して ID フラグの値を読み取り、POPFD 命令を使用して ID フラグの値を変更する方法を示している。

1. 一部の Intel486 プロセッサおよびそれ以降のプロセッサのみ。Intel386 プロセッサの EFLAGS レジスタのビット 21 は、ソフトウェアによって変更できないので、Intel386 プロセッサは CPUID 命令を実行できない。CPUID 命令をサポートしていないプロセッサ上で CPUID 命令を実行すると、無効オペコード例外が発生する。

3 CPUID 命令の出力

CPUID 命令は、2 つの機能セットをサポートしている。第 1 の機能セットは、基本プロセッサ情報を返す。第 2 の機能セットは、拡張プロセッサ情報を返す。図 2 は、CPUID 命令によって出力される基本プロセッサ情報についてまとめたものである。CPUID 命令の出力は、EAX レジスタの内容に完全に依存している。つまり、EAX レジスタの値を変更して CPUID を実行すると、CPUID 命令は、EAX レジスタ内の値に基づいて特定の機能を実行する（表 1 を参照）。基本プロセッサ情報を返す CPUID 機能について、EAX レジスタに入力可能な最大値を確認するには、プログラムは、次のように、EAX レジスタのパラメータ値を "0" に設定し、CPUID 命令を実行する。

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

この CPUID 命令の実行後、EAX レジスタに戻り値が格納される。EAX のパラメータ値には、0 以上で、この EAX の最大戻り値以下の値を常に使用すること。

拡張プロセッサ情報を返す CPUID 機能について、EAX レジスタに入力可能な最大値を確認するには、プログラムは、次のように、EAX レジスタのパラメータ値を "80000000h" に設定し、CPUID 命令を実行する。

```
MOV  EAX, 80000000H
CPUID
```

この CPUID 命令の実行後、EAX レジスタに戻り値が格納される。EAX パラメータ値には、80000000h 以上で、この EAX の最大戻り値以下の値を常に使用すること。現在および将来の IA-32 プロセッサでは、いずれの機能セットでも、最大値を超える入力パラメータを指定して CPUID を実行した場合や、拡張機能がサポートされていない場合は、EAX レジスタのビット 31 がクリアされる。EAX をプロセッサの最大値より大きな値に設定して CPUID 命令を実行した場合、プロセッサが返す（ビット 31 以外の）すべてのビット値はモデル固有であり、設計の前提にしてはならない。

3.1 ベンダ ID ストリング

EAX レジスタに返される最大値以外に、インテル・ベンダ ID ストリングも同時に確認できる。EAX レジスタの入力値が 0 の場合、CPUID 命令は、EBX、EDX、ECX レジスタにベンダ識別ストリングも返す（図 2 を参照）。これらのレジスタには、次の ASCII ストリングが返される。

GenuineIntel

インテル® アーキテクチャと互換性を持つ製品を有する他社は、CPUID 命令を実装できるが、自社の部品がインテル純正部品であると合法的に主張することはできない。したがって、"GenuineIntel" ストリングが返された場合は、CPUID 命令とプロセッサ・シグニチャが本書で説明したとおりに実装されていることが保証される。CPUID 命令の実行後に "GenuineIntel" ストリングが返されない場合は、CPUID 命令によって返される情報を解釈する際に、本書で説明した内容を前提にしてはならない。

図 2. CPUID 命令の出力

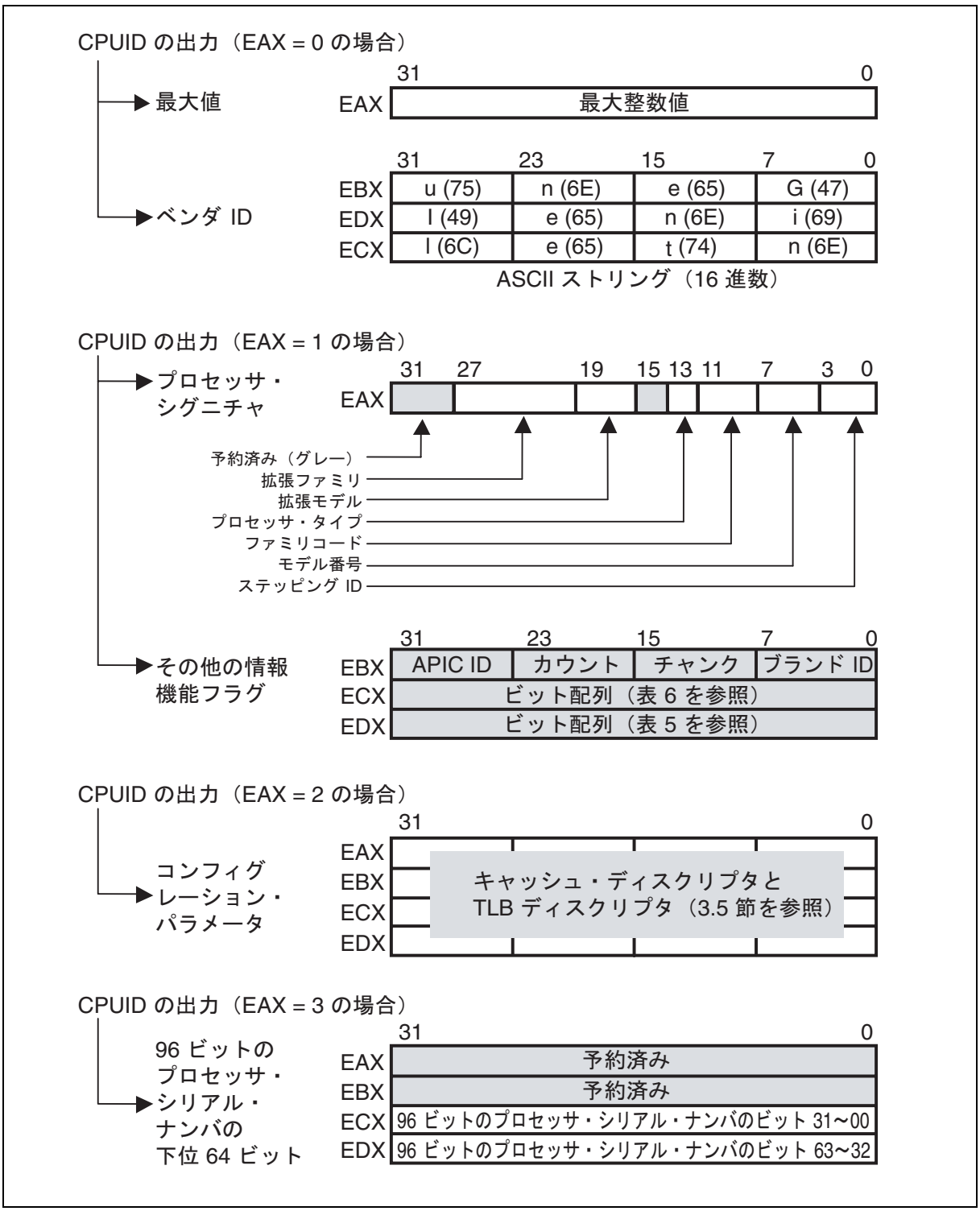


表 1. CPUID 命令によって返される情報

EAX の初期値	プロセッサに関して提供される情報
	基本 CPUID 情報
0H	EAX： 最大入力値（基本 CPUID 情報の場合） EBX： "Genu" ECX： "ntel" EDX： "inel"
1H	EAX： **32 ビットのプロセッサ・シグニチャ（拡張ファミリ、拡張モデル、タイプ、ファミリ、モデル、ステッピング ID）。**PSN 機能フラグがセットされている場合は、96 ビットのプロセッサ・シリアル・ナンバのビット 95 ～ 64 が返される。 EBX： ビット 7 ～ 0： ブランド・インデックス - 値 = 00h の場合はサポートされない。 ビット 15 ～ 8： CLFLUSH ラインサイズ（戻り値 * 8 = キャッシュ・ライン・サイズ）**CLFSH 機能フラグがセットされている場合にのみ有効。** ビット 23 ～ 16： 論理プロセッサの数 **ハイパー・スレッディング・テクノロジ・フラグがセットされている場合にのみ有効。** ビット 31 ～ 24： プロセッサのローカル APIC の物理 ID **Pentium® 4 プロセッサおよびそれ以降のプロセッサで有効。** ECX： 機能フラグ（表 6 を参照） EDX： 機能フラグ（表 5 を参照）
2H	EAX、EBX、ECX、EDX　キャッシュ・ディスクリプタと TLB ディスクリプタ
3H	EAX： 予約済み EBX： 予約済み ECX： ビット 31 ～ 0 は、96 ビットのプロセッサ・シリアル・ナンバのビット 31 ～ 0 を表す（Pentium® III プロセッサで PSN 機能フラグがセットされている場合にのみ有効。それ以外の場合は、このレジスタの値は予約済みである）。 EDX： ビット 31 ～ 0 は、96 ビットのプロセッサ・シリアル・ナンバのビット 63 ～ 32 を表す（Pentium III プロセッサで PSN 機能フラグがセットされている場合にのみ有効。それ以外の場合は、このレジスタの値は予約済みである）。 **注：** プロセッサ・シリアル・ナンバ（PSN）は、Pentium® 4 プロセッサおよびそれ以降のプロセッサではサポートしていない。すべてのモデルで、PSN 機能を使用する前に、（CPUID を使用して返される）PSN フラグを使用して PSN のサポートをチェックすること。PSN 機能フラグの値が "0" の場合は、プロセッサ・シリアル・ナンバ機能がサポートされていないか、Pentium III プロセッサ上で無効にされている。詳細については、4.1 節を参照のこと。

表 1. CPUID 命令によって返される情報

EAX の初期値	プロセッサに関して提供される情報（続き）
	基本 CPUID 情報
4H	**決定性キャッシュ・パラメータ機能** EAX： ビット 4 ～ 0： キャッシュ・タイプ ** ビット 7 ～ 5： キャッシュ・レベル（1 から始まる） ビット 8： 自己初期化するキャッシュ・レベル（ソフトウェアによる初期化を必要としない） ビット 9： フル・アソシアティブ・キャッシュ ビット 13 ～ 10： 予約済み ビット 25 ～ 14： このキャッシュを共有するスレッドの数 * ビット 31 ～ 26： ダイ上のプロセッサ・コアの数（マルチコア）* EBX： ビット 11 ～ 0： L = システム・コヒーレンシ・ライン・サイズ * ビット 21 ～ 12： P = 物理ライン・パーティション * ビット 31 ～ 22： W = アソシアティブ・ウェイ数 * ECX： ビット 31 ～ 0： S = セット数 * EDX： 予約済み * 値を求めるには、レジスタファイル内の値に 1 を加算する。例えば、プロセッサ・コアの数は EAX[31：26]+1 である。 ** キャッシュ・タイプ・フィールド 0 = Null - キャッシュなし 1 = データ・キャッシュ 2 = 命令キャッシュ 3 = ユニファイド・キャッシュ 31 ～ 4 = 予約済み **注：** 決定性キャッシュ・パラメータ機能は、IA32_MISC_ENABLES.LIMIT_CPUID_MAXVAL（ビット 22）が '0' にクリアされている場合にのみ利用可能である（デフォルト）。
5H	**MONITOR/MWAIT 機能** EAX： ビット 15 ～ 0： 最小モニタ・ライン・サイズ（バイト単位）（デフォルトはプロセッサのモニタ・グラニュラリティ） ビット 31 ～ 16： 予約済み EBX： ビット 15 ～ 0： 最大モニタ・ライン・サイズ（バイト単位）（デフォルトはプロセッサのモニタ・グラニュラリティ） ビット 31 ～ 16： 予約済み ECX： 予約済み EDX： 予約済み **注：** MONITOR/MWAIT 機能は、IA32_MISC_ENABLES. LIMIT_MAXVAL（ビット 22）が '0' にクリアされている場合にのみ利用可能である（デフォルト）。
80000000H	EAX： 最大入力値（拡張機能 CPUID 情報の場合） EBX、ECX、EDX：予約済み
80000001H	EAX、EBX：追加の拡張機能フラグ用に予約済み ECX、EDX：拡張機能フラグ（表 7 と表 8 を参照）

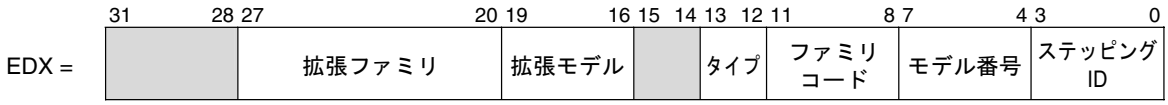
表 1. CPUID 命令によって返される情報

EAX の初期値	プロセッサに関して提供される情報（続き）
	拡張機能 CPUID 情報
80000002H	EAX： プロセッサ・ブランド・ストリング EBX、ECX、EDX：プロセッサ・ブランド・ストリング（続き）
80000003H	EAX、EBX、ECX、EDX：プロセッサ・ブランド・ストリング（続き）
80000004H	EAX、EBX、ECX、EDX：プロセッサ・ブランド・ストリング（続き）
80000005h	EAX、EBX、ECX、EDX：予約済み
80000006h	**拡張 L2 キャッシュ機能** EAX： 予約済み EBX： 予約済み ECX： ビット 7～0： L2 キャッシュ・ライン・サイズ ビット 11 ～8： 予約済み ビット 15 ～12： L2 キャッシュ・アソシアティビティ ビット 31 ～16： L2 キャッシュ・サイズ（1K 単位） EDX： 予約済み
80000007h	EAX、EBX、ECX、EDX：予約済み
80000008h	**拡張アドレスサイズ機能** EAX： ビット 7～0： 物理アドレスサイズ（ビット数） ビット 15 ～8： 仮想アドレスサイズ（ビット数） ビット 31 ～16： 予約済み EBX、ECX、EDX：予約済み

3.2 プロセッサ・シグニチャ

Intel486™ プロセッサ・ファミリから、リセット後にはEDX レジスタにプロセッサ識別シグニチャが格納される（図 3 を参照）。**プロセッサ識別シグニチャは 32 ビット値である。**プロセッサ・シグニチャは、8 つの異なるビット・フィールドで構成される。グレーのフィールドは予約ビットであり、プロセッサ・シグニチャを利用する際はマスクアウトする必要がある。プロセッサ識別シグニチャは、そのほかの 6 つのフィールドで構成される。

図 3. RESET 後の EDX レジスタ



CPUID 命令を実装しているプロセッサも、リセット後に 32 ビットのプロセッサ識別シグニチャを返す。ただし、CPUID 命令を使用すれば、プロセッサ・シグニチャをいつでも確認できる。図 3 は、Intel486 プロセッサとそれ以降のインテル・プロセッサの 32 ビット・プロセッサ・シグニチャのフォーマットを示している。なお、リセット後の EDX レジスタのプロセッサ・シグニチャ値は、図 2 の EAX レジスタのプロセッサ・シグニチャ出力値と同じになる。表 4 は、Intel486 プロセッサとそれ以降のプロセッサの現在定義されている EAX レジスタの戻り値を示している。

拡張ファミリ（ビット位置 20 ～ 27）とファミリコード（ビット位置 8 ～ 11）を組み合わせて、プロセッサが Intel386™ プロセッサ、Intel486 プロセッサ、Pentium® プロセッサ、Pentium® Pro プロセッサ、Pentium® 4 プロセッサのうちどのプロセッサ・ファミリに属しているかを示す。P6 ファミリ・プロセッサには、Pentium Pro プロセッサ・アーキテクチャ・ベースのすべてのプロセッサが含まれ、拡張ファミリの値は 00h、ファミリコードの値は 6h である。Pentium 4 プロセッサ・ファミリには、Intel NetBurst® マイクロアーキテクチャ・ベースのすべてのプロセッサが含まれ、拡張ファミリの値は 00h、ファミリコードの値は 0Fh である。

拡張モデル（ビット位置 16 ～ 19）とモデル番号（ビット 4 ～ 7）を組み合わせて、プロセッサ・ファミリ内のプロセッサのモデルを特定できる。ビット 0 ～ 3 のステッピング ID は、当該モデルのリビジョン番号を示す。

プロセッサ・タイプ（表 2 のビット位置 12 と 13）は、プロセッサがオリジナル OEM プロセッサ、OverDrive® プロセッサ、デュアル・プロセッサ（デュアル・プロセッサ・システムで使用可能）のいずれであるかを示す。表 2 は、EAX レジスタのビット 12 と 13 に返されるプロセッサ・タイプの値を示している。

表 2. プロセッサ・タイプ（ビット位置 13 と 12）

値	説明
00	オリジナル OEM プロセッサ
01	OverDrive プロセッサ
10	デュアル・プロセッサ
11	インテルで予約済み（使用不可）

Pentium® II プロセッサ（モデル 5）、Pentium® II Xeon™ プロセッサ（モデル 5）、Celeron® プロセッサ（モデル 5）は、同じ拡張ファミリ、ファミリコード、拡張モデル、モデル番号を共有している。これらのプロセッサを区別するには、ソフトウェアは、EAX = 2 に設定して CPUID 命令を実行し、キャッシュ・ディスクリプタの値をチェックする必要がある。L2 キャッシュが返されない場合は、プロセッサはインテル Celeron プロセッサ（モデル 5）として特定される。1MB または 2MB の L2 キャッシュ・サイズが返される場合は、そのプロセッサは Pentium II Xeon プロセッサである。それ以外の場合は、Pentium II プロセッサ（モデル 5）、または 512KB L2 キャッシュを搭載した Pentium II Xeon プロセッサである。

Pentium III プロセッサ（モデル 7）と Pentium® III Xeon™ プロセッサ（モデル 7）は、同じ拡張ファミリ、ファミリコード、拡張モデル、モデル番号を共有している。これらのプロセッサを区別するには、ソフトウェアは、EAX = 2 に設定して CPUID 命令を実行し、キャッシュ・ディスクリプタの値をチェックする必要がある。1M または 2M の L2 キャッシュ・サイズが返される場合は、そのプロセッサは Pentium III Xeon プロセッサである。それ以外の場合は、Pentium III プロセッサ、または 512KB L2 キャッシュを搭載した Pentium III Xeon プロセッサである。

Pentium III プロセッサ（モデル 8）、Pentium III Xeon プロセッサ（モデル 8）、Celeron プロセッサ（モデル 8）のプロセッサ・ブランドは、EAX = 1 に設定して CPUID 命令を実行した場合に返されるブランド ID の値によって確認できる。表 12 は、ブランド ID によって定義されるプロセッサ・ブランドを示している。

Intel486 SX プロセッサ、Intel486 DX プロセッサ、IntelDX2™ プロセッサの古いバージョンは、CPUID 命令をサポートしていない。[2] したがって、これらのプロセッサは、リセット時にのみプロセッサ・シグニチャを返す。プロセッサごとの CPUID 命令の対応状況については、表 4 を参照のこと。

図 4 は、Intel386 プロセッサのプロセッサ・シグニチャのフォーマットを示している。このフォーマットは、他のプロセッサとは異なる。表 3 は、Intel386 プロセッサの現在定義されているプロセッサ・シグニチャの値を示している。

図 4. Intel386™ プロセッサのプロセッサ・シグニチャのフォーマット

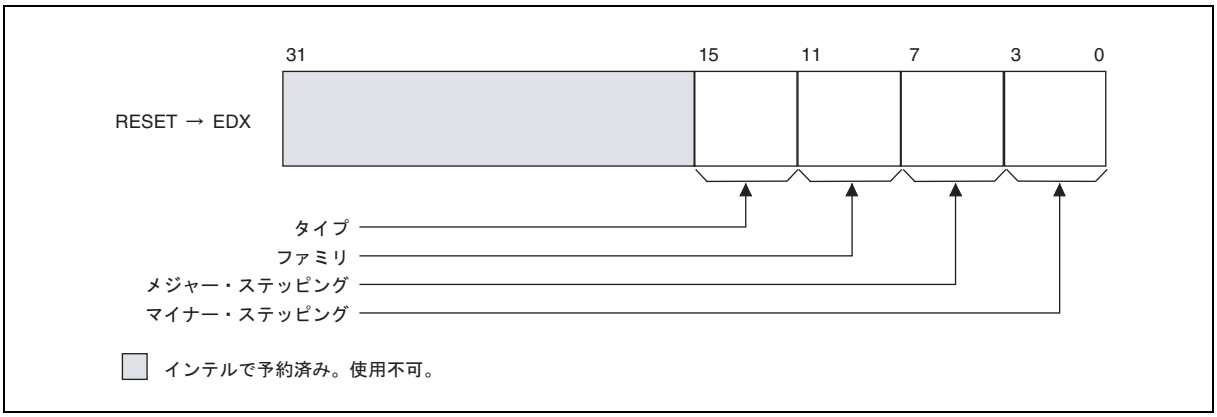


表 3. Intel386™ プロセッサのプロセッサ・シグニチャ

タイプ	ファミリ	メジャー・ステッピング	マイナー・ステッピング	説明
0000	0011	0000	xxxx	Intel386 DX プロセッサ
0010	0011	0000	xxxx	Intel386 SX プロセッサ
0010	0011	0000	xxxx	Intel386 CX プロセッサ
0010	0011	0000	xxxx	Intel386 EX プロセッサ
0100	0011	0000 および 0001	xxxx	Intel386 SL プロセッサ
0000	0011	0100	xxxx	RapidCAD* コプロセッサ

表 4. Intel486™ プロセッサおよびそれ以降のプロセッサのプロセッサ・シグニチャ

拡張ファミリ	拡張モデル	タイプ	ファミリコード	モデル番号	ステッピング ID	説明
00000000	0000	00	0100	000x	xxxx (1)	Intel486 DX プロセッサ
00000000	0000	00	0100	0010	xxxx (1)	Intel486 SX プロセッサ
00000000	0000	00	0100	0011	xxxx (1)	Intel487 プロセッサ
00000000	0000	00	0100	0011	xxxx (1)	IntelDX2 プロセッサ
00000000	0000	00	0100	0011	xxxx (1)	IntelDX2™ OverDrive® プロセッサ
00000000	0000	00	0100	0100	xxxx (3)	Intel486 SL プロセッサ

2. すべての Intel486 SL エンハンスト・プロセッサおよびライトバック・エンハンスト・プロセッサは、CPUID 命令を実行できる。表 4 を参照のこと。

表 4. Intel486™ プロセッサおよびそれ以降のプロセッサのプロセッサ・シグニチャ（続き）

拡張ファミリ	拡張モデル	タイプ	ファミリコード	モデル番号	ステッピング ID	説明
00000000	0000	00	0100	0101	xxxx (1)	IntelSX2™ プロセッサ
00000000	0000	00	0100	0111	xxxx (3)	ライトバック・エンハンスト IntelDX2 プロセッサ
00000000	0000	00	0100	1000	xxxx (3)	IntelDX4™ プロセッサ
00000000	0000	0x	0100	1000	xxxx (3)	IntelDX4™ OverDrive® プロセッサ
00000000	0000	00	0101	0001	xxxx (2)	Pentium プロセッサ（60、66）
00000000	0000	00	0101	0010	xxxx (2)	Pentium プロセッサ（75、90、100、120、133、150、166、200）
00000000	0000	01 (4)	0101	0001	xxxx (2)	Pentium プロセッサ（60、66）用 Pentium® OverDrive® プロセッサ
00000000	0000	01 (4)	0101	0010	xxxx (2)	Pentium プロセッサ（75、90、100、120、133）用 Pentium OverDrive プロセッサ
00000000	0000	01	0101	0011	xxxx (2)	Intel486 プロセッサ・ベース・システム用 Pentium OverDrive プロセッサ
00000000	0000	00	0101	0100	xxxx (2)	MMX® テクノロジ Pentium® プロセッサ（166、200）
00000000	0000	01	0101	0100	xxxx (2)	Pentium プロセッサ（75、90、100、120、133）用 MMX® テクノロジ Pentium® OverDrive® プロセッサ
00000000	0000	00	0110	0001	xxxx (2)	Pentium® Pro プロセッサ
00000000	0000	00	0110	0011	xxxx (2)	Pentium II プロセッサ（モデル 3）
00000000	0000	00	0110	0101(5)	xxxx (2)	Pentium II プロセッサ（モデル 5）、Pentium II Xeon プロセッサ（モデル 5）、インテル Celeron プロセッサ（モデル 5）
00000000	0000	00	0110	0110	xxxx (2)	Celeron プロセッサ（モデル 6）
00000000	0000	00	0110	0111(6)	xxxx (2)	Pentium III プロセッサ（モデル 7）、Pentium III Xeon プロセッサ（モデル 7）
00000000	0000	00	0110	1000(7)	xxxx (2)	Pentium III プロセッサ（モデル 8）、Pentium III Xeon プロセッサ（モデル 8）、Celeron プロセッサ（モデル 8）
00000000	0000	00	0110	1001	xxxx (2)	インテル® Pentium® M プロセッサ（モデル 9）
00000000	0000	00	0110	1010	xxxx (2)	Pentium III Xeon プロセッサ（モデル A）
00000000	0000	00	0110	1011	xxxx (2)	Pentium III プロセッサ（モデル B）
00000000	0000	00	0110	1101	xxxx (2)	インテル Pentium M プロセッサ（モデル D）。すべてのプロセッサは 90nm プロセスを使用して製造。
00000000	0000	01	0110	0011	xxxx (2)	インテル® Pentium® II OverDrive® プロセッサ
00000000	0000	00	1111	0000	xxxx (2)	Pentium® 4 プロセッサ、インテル Xeon プロセッサ。すべてのプロセッサはモデル 0 であり、0.18 ミクロン・プロセスを使用して製造。

表 4. Intel486™ プロセッサおよびそれ以降のプロセッサのプロセッサ・シグニチャ（続き）

拡張ファミリ	拡張モデル	タイプ	ファミリコード	モデル番号	ステッピング ID	説明
00000000	0000	00	1111	0001	xxxx (2)	Pentium 4 プロセッサ、インテル Xeon プロセッサ、インテル® Xeon™ プロセッサ MP、インテル Celeron プロセッサ。すべてのプロセッサはモデル 1 であり、0.18 ミクロン・プロセスを使用して製造。
00000000	0000	00	1111	0010	xxxx (2)	Pentium 4 プロセッサ、モバイル インテル® Pentium® 4 プロセッサ - M、インテル Xeon プロセッサ、インテル Xeon プロセッサ MP、インテル Celeron プロセッサ、モバイル インテル® Celeron® プロセッサ。すべてのプロセッサはモデル 2 であり、0.13 ミクロン・プロセスを使用して製造。
00000000	0000	00	1111	0011	xxxx (2)	Pentium 4 プロセッサ、モバイル インテル Pentium 4 プロセッサ - M、インテル Xeon プロセッサ、インテル Celeron プロセッサ。すべてのプロセッサはモデル 3 であり、90nm プロセスを使用して製造。
00000000	0000	00	1111	0100	xxxx (2)	Pentium 4 プロセッサ、モバイル インテル Pentium 4 プロセッサ - M、インテル Xeon プロセッサ、インテル Celeron プロセッサ。すべてのプロセッサはモデル 4 であり、90nm プロセスを使用して製造。

注：

1. このプロセッサは CPUID 命令を実装していない。
2. ステッピング番号の最新のリストについては、Intel486™ の資料、『Pentium® Processor Specification Update』（資料番号 242480）、『Pentium® Pro Processor Specification Update』（資料番号 242689）、『Pentium® II Processor Specification Update』（資料番号 243337）、『Pentium® II Xeon™ Processor Specification Update』（資料番号 243776）、『Intel® Celeron® Processor Specification Update』（資料番号 243748）、『Pentium® III Processor Specification Update』（資料番号 244453）、『Pentium® III Xeon™ Processor Specification Update』（資料番号 244460）、『Pentium® 4 Processor Specification Update』（資料番号 249199）、『Intel® Xeon™ Processor Specification Update』（資料番号 249678）、または『Intel® Xeon™ Processor MP Specification Update』（資料番号 290741）を参照のこと。
3. ステッピング 3 は CPUID 命令を実装している。
4. OverDrive プロセッサのタイプ・フィールドの定義は 01h である。Pentium OverDrive プロセッサのエラッタのため、タイプとして常に 00h が返される。
5. Pentium II プロセッサ（モデル 5）、Pentium II Xeon プロセッサ（モデル 5）、Celeron プロセッサ（モデル 5）を区別するには、ソフトウェアは、EAX = 2 に設定して CPUID 命令を実行し、キャッシュ・ディスクリプタの値をチェックする必要がある。L2 キャッシュが返されない場合は、プロセッサは Celeron プロセッサ（モデル 5）として特定される。1M または 2M の L2 キャッシュ・サイズが返される場合は、そのプロセッサは Pentium II Xeon プロセッサである。それ以外の場合は、Pentium II プロセッサ（モデル 5）、または 512KB L2 キャッシュを搭載した Pentium II Xeon プロセッサである。
6. Pentium III プロセッサ（モデル 7）と Pentium III Xeon プロセッサ（モデル 7）を区別するには、ソフトウェアは、EAX = 2 に設定して CPUID 命令を実行し、キャッシュ・ディスクリプタの値をチェックする必要がある。1M または 2M の L2 キャッシュ・サイズが返される場合は、そのプロセッサは Pentium III Xeon プロセッサである。それ以外の場合は、Pentium III プロセッサ、または 512KB L2 キャッシュを搭載した Pentium III Xeon プロセッサである。
7. Pentium III プロセッサ（モデル 8）と Pentium III Xeon プロセッサ（モデル 8）を区別するには、ソフトウェアは、EAX = 1 に設定して CPUID 命令を実行し、ブランド ID の値をチェックする必要がある。

3.3 機能フラグ

EAX レジスタ内の値が 1 の場合、CPUID 命令は（EAX レジスタにプロセッサ・シグニチャをロードする以外に）EDX レジスタと ECX レジスタに機能フラグをロードする。これらの機能フラグは（フラグ = 1 の場合）、プロセッサがサポートしている機能を示す。表 5 と表 6 は、現在定義されている機能フラグの値を示している。

将来のプロセッサについては、プログラマーズ・リファレンス・マニュアル、ユーザーズ・マニュアル、または最新の機能フラグの値に関する適切な資料を参照のこと。

注意：アプリケーション内で機能フラグを使用して、サポートされているプロセッサ機能を確認すること。ソフトウェアは、CPUID 機能フラグを使用してプロセッサ機能を確認することにより、プロセッサ機能の追加や削除によって互換性が失われる状態を検出し、回避できる。

表 5. EDX レジスタに返される機能フラグの値

ビット	名前	説明（フラグ = 1 の場合）	注釈
0	FPU	オンチップの浮動小数点ユニット	プロセッサは Intel387™ プロセッサ浮動小数点命令セットをサポートする FPU を搭載している。
1	VME	仮想モード拡張機能	プロセッサは仮想 8086 モード拡張機能をサポートしている。
2	DE	デバッグ拡張機能	プロセッサは I/O ブレークポイントをサポートしている。これには、デバッグ拡張機能と DR4/DR5 レジスタへのアクセスのトラップ機能（オプション）をイネーブルにするための CR4.DE ビットが含まれる。
3	PSE	ページサイズ拡張機構	プロセッサは 4M バイトページをサポートしている。
4	TSC	タイムスタンプ・カウンタ	RDTSC 命令をサポートしている。これには、アクセス / 特権制御用の CR4.TSD ビットが含まれる。
5	MSR	モデル固有レジスタ	モデル固有レジスタが、RDMSR 命令、WRMSR 命令と一緒に実装されている。
6	PAE	物理アドレス拡張機構	32 ビットを超える物理アドレスをサポートしている。
7	MCE	マシンチェック例外	マシンチェック例外（例外 18）と CR4.MCE イネーブル・ビットをサポートしている。
8	CX8	CMPXCHG8 命令のサポート	8 バイト比較交換命令をサポートしている。
9	APIC	オンチップ APIC ハードウェアのサポート	プロセッサはソフトウェアによるアクセスが可能なローカル APIC を搭載している。
10		予約済み	この値を設計の前提にしてはならない。
11	SEP	高速システムコール	プロセッサが高速システムコール命令 SYSENTER と SYSEXIT をサポートしているかを示す。 **注：**SYSENTER/SYSEXIT 機能と SEP 機能ビットについての詳細は、3.4 節を参照のこと。
12	MTRR	メモリタイプ範囲レジスタ	プロセッサは、メモリタイプ範囲レジスタ（厳密には、MTRR_CAP レジスタ）をサポートしている。
13	PGE	ページ・グローバル・イネーブル	ページ・ディレクトリ・エントリ（PDE）内とページ・テーブル・エントリ（PTE）内でグローバル・ビットをサポートしている。グローバル・ビットは、異なるプロセスに共通であるためフラッシュする必要がない TLB エントリを示す。この機能は CR4.PGE ビットによって制御される。

表 5. EDX レジスタに返される機能フラグの値（続き）

ビット	名前	説明（フラグ = 1 の場合）	注釈
14	MCA	マシン・チェック・アーキテクチャ	マシン・チェック・アーキテクチャ（厳密には、MCG_CAP レジスタ）をサポートしている。
15	CMOV	条件付き移動命令のサポート	プロセッサは CMOVcc をサポートしている。FPU 機能フラグ（ビット 0）もセットされている場合は、FCMOVCC 命令と FCOMI 命令もサポートしている。
16	PAT	ページ属性テーブル	プロセッサがページ属性テーブル（PAT）をサポートしているかを示す。この機能は、メモリタイプ範囲レジスタ（MTRR）を補強し、オペレーティング・システムがリニアアドレスの 4K グラニュラリティでメモリの属性を指定できるようにする。
17	PSE-36	36 ビット・ページ・サイズ拡張機構	プロセッサが、4GB を超える物理メモリのアドレス指定ができる 4M バイトページをサポートしているかを示す。この機能は、4M バイトページの物理アドレスの上位 4 ビットが、ページ・ディレクトリ・エントリのビット 13 ～ 16 によってコード化されることを示す。
18	PSN	プロセッサ・シリアル・ナンバが存在し、イネーブルになっている	プロセッサは 96 ビットのプロセッサ・シリアル・ナンバ機能をサポートしており、その機能がイネーブルになっている。 **注：**Pentium 4 プロセッサ・ファミリのプロセッサは、この機能をサポートしていない。
19	CLFSH	CLFLUSH 命令のサポート	プロセッサが CLFLUSH 命令をサポートしていることを示す。
20		予約済み	この値を設計の前提にしてはならない。
21	DS	デバッグストア	プロセッサが、アドレスへの分岐とアドレスからの分岐の履歴をメモリバッファに書き込む機能を持っていることを示す。
22	ACPI	温度モニタとソフトウェア制御クロック機能のサポート	プロセッサは、プロセッサの温度を監視し、ソフトウェアの制御下であらかじめ定義されたデューティ・サイクルでプロセッサのパフォーマンスを調整するための、内部 MSR を実装している。
23	MMX	インテル・アーキテクチャ MMX テクノロジのサポート	プロセッサはインテル・アーキテクチャ MMX テクノロジ拡張命令セットをサポートしている。
24	FXSR	高速浮動小数点セーブ / リストア	プロセッサが浮動小数点コンテキストの高速セーブ / リストア用の FXSAVE 命令と FXRSTOR 命令をサポートしているかを示す。このビットがセットされている場合は、オペレーティング・システムは CR4.OSFXSR を利用して、オペレーティング・システムが高速セーブ / リストア命令を使用することを指示できる。
25	SSE	ストリーミング SIMD 拡張命令のサポート	プロセッサはインテル・アーキテクチャ・ストリーミング SIMD 拡張命令をサポートしている。
26	SSE2	ストリーミング SIMD 拡張命令 2	プロセッサはストリーミング SIMD 拡張命令 2 をサポートしている。
27	SS	自己スヌープ	プロセッサは、バスに発行されるトランザクションについてプロセッサのキャッシュ構造のスヌープを実行することで、競合するメモリタイプを管理する機能をサポートしている。

表 5. EDX レジスタに返される機能フラグの値（続き）

ビット	名前	説明（フラグ = 1 の場合）	注釈
28	HTT	ハイパー・スレッディング・テクノロジ†	このプロセッサのマイクロアーキテクチャは、同一の物理パッケージ内で複数の論理プロセッサを動作させる機能を持っている。 このフィールドは、このプロセッサのハイパー・スレッディング・テクノロジがイネーブルになっていることを示すわけではない。ハイパー・スレッディング・テクノロジをサポートしているかを確認するには、EAX=1 に設定して CPUID を実行した後、EBX[23：16] に返される値をチェックする。EBX[23：16] 内の値が 1 より大きい場合は、プロセッサはハイパー・スレッディング・テクノロジをサポートしている。
29	TM	温度モニタのサポート	プロセッサは、温度モニタの自動温度制御回路（TCC）を実装している。
30		予約済み	この値を設計の前提にしてはならない。
31	PBE	ペンディング・ブレーク・イネーブル	プロセッサは、プロセッサがストップクロック・ステートになっている（STPCLK# がアサートされている）とき、割り込みが未処理であり、通常の動作に戻って割り込みを処理する必要があることをプロセッサに報告する、FERR#/PBE# ピンの使用をサポートしている。この機能は、IA32_MISC_ENABLE MSR 内のビット 10（PBE イネーブル）によってイネーブルにされる。

表 6. ECX レジスタに返される機能フラグの値

ビット	名前	説明（フラグ = 1 の場合）	注釈
0	SSE3	ストリーミング SIMD 拡張命令 3	プロセッサはストリーミング SIMD 拡張命令 3 をサポートしている。
2：1		予約済み	この値を設計の前提にしてはならない。
3	MONITOR	MONITOR/MWAIT	プロセッサは MONITOR 命令と MWAIT 命令をサポートしている。
4	DS-CPL	CPL の条件を満たすデバッグストア	プロセッサは、CPL の条件を満たす分岐メッセージの格納を可能にする、デバッグストア拡張機能をサポートしている。
6：5		予約済み	この値を設計の前提にしてはならない。
7	EST	拡張版 Intel SpeedStep® テクノロジ	プロセッサは第二世代の Intel SpeedStep テクノロジを実装している。
8	TM2	温度モニタ 2	プロセッサは、温度モニタ 2 の温度制御回路（TCC）を実装している。
9		予約済み	この値を設計の前提にしてはならない。
10	CID	コンテキスト ID	BIOS によって、L1 データ・キャッシュ・モードをアダプティブ・モードまたは共有モードに設定できる。
12：11		予約済み	この値を設計の前提にしてはならない。
13	CX16	CMPXCHG16B	このプロセッサは CMPXCHG16B 命令をサポートしている。
14	xTPR	タスク・プライオリティ・メッセージの送信	プロセッサは、タスク・プライオリティ・メッセージの送信をディスエーブルにする機能をサポートしている。この機能フラグがセットされている場合は、タスク・プライオリティ・メッセージをディスエーブルにできる。タスク・プライオリティ・メッセージの送信は、IA32_MISC_ENABLE MSR 内のビット 23（エコー TPR ディスエーブル）によって制御される。
31：15		予約済み	この値を設計の前提にしてはならない。

3.4 拡張機能フラグ

EAX レジスタ内の値が 80000001h の場合、CPUID 命令は、EDX レジスタに拡張機能フラグをロードする。この機能フラグは（フラグ = 1 の場合）、プロセッサがサポートしている拡張機能を示す。表 7 は、現在定義されている機能フラグの値を示している。

将来のプロセッサについては、プログラマーズ・リファレンス・マニュアル、ユーザーズ・マニュアル、または最新の拡張機能フラグの値に関する適切な資料を参照のこと。

注意：アプリケーション内で拡張機能フラグを使用して、サポートされているプロセッサ機能を確認すること。ソフトウェアは、CPUID 機能フラグを使用してプロセッサ機能を確認することにより、プロセッサ機能の追加や削除によって互換性が失われる状態を検出し、回避できる。

表 7. EDX レジスタに返される拡張機能フラグの値

ビット	名前	説明（フラグ = 1 の場合）	注釈
10：0		予約済み	この値を設計の前提にしてはならない。
11	SYSCALL	SYSCALL/SYSRET	プロセッサは SYSCALL 命令と SYSRET 命令をサポートしている。
19：12		予約済み	この値を設計の前提にしてはならない。
20	XD Bit	エグゼキュート・ディスエーブル・ビット	プロセッサは、PAE モード・ページングがイネーブルの場合、XD ビットをサポートする。
28：21		予約済み	この値を設計の前提にしてはならない。
29	インテル EM64T	インテル エクステンデッド・メモリ 64 テクノロジ	プロセッサは IA-32 アーキテクチャの 64 ビット拡張技術をサポートしている。詳細については、『64-bit Extensions Technology Software Developers Guide』（資料番号 300834、300835）を参照のこと。この資料は、http://developer.intel.com/technology/64bitextensions/（英語）で入手できる。
31：30		予約済み	この値を設計の前提にしてはならない。

表 8. ECX レジスタに返される拡張機能フラグの値

ビット	名前	説明（フラグ = 1 の場合）	注釈
19：0		予約済み	この値を設計の前提にしてはならない。
20	LAHF	LAHF / SAHF	値が 1 の場合は、LAHF 命令と SAHF 命令が利用可能であることを示す（IA-32e モードがイネーブルであり、プロセッサが 64 ビット・サブモードで動作している場合）。詳細については、『IA-32e BIOS Writer's Guide』を参照のこと。
31：21		予約済み	この値を設計の前提にしてはならない。

3.5 SYSENTER/SYSEXIT - SEP 機能ビット

CPUID の SYSENTER 存在（SEP）ビット 11 は、SYSENTER 機能の存在を示す。SEP ビットの状態をチェックするオペレーティング・システムは、プロセッサ・ファミリとモデルもチェックして、SYSENTER/SYSEXIT 命令が実際にサポートされていることを確認しなければならない。

```
IF (CPUID SEP bit is set)
{
   IF ((Processor Signature & 0x0FFF3FFF) < 0x00000633)
      Fast System Call is NOT supported
   ELSE
      Fast System Call is supported
}
```

Pentium Pro プロセッサ（モデル = 1）は、CPUID 命令に対して、セットされた SEP 機能ビットを返すが、ソフトウェアはこの機能を使用できない。

3.6 キャッシュ・サイズ、フォーマット、TLB の情報

EAX レジスタ内の値が 2 の場合、CPUID 命令は、EAX、EBX、ECX、EDX レジスタに、プロセッサのキャッシュと TLB の特性を示すディスクリプタをロードする。EAX レジスタの下位 8 ビット（AL）は、プロセッサのキャッシュ・システムの全体像を得るまでに CPUID を何回実行しなければならないかを示す値を格納する。例えば、Pentium 4 プロセッサでは、EAX レジスタの下位 8 ビットの戻り値は 1 になる。これは、CPUID 命令を（EAX = 2 に設定して）1 回だけ実行すれば、プロセッサ構成の全体像を得られることを示している。

EAX レジスタの下位 8 ビット以外のビット、EBX レジスタ、ECX レジスタ、EDX レジスタには、キャッシュ・ディスクリプタと TLB ディスクリプタが返される。表 9 に示すように、特定のレジスタのビット 31 が 0 の場合、そのレジスタには有効な 8 ビット・ディスクリプタが格納されている。ディスクリプタをデコードするには、レジスタの最上位バイトからレジスタの最下位バイトに順次移動する。ビット 31 が 0 であるとすると、そのレジスタは、ビット 24 ～ 31、ビット 16 ～ 23、ビット 8 ～ 15、ビット 0 ～ 7 に、有効なキャッシュ・ディスクリプタまたは TLB ディスクリプタを格納している。ソフトウェアは、各ディスクリプタ・ビット・フィールド内の値と表 10 に示した値を比較し、プロセッサのキャッシュと TLB の特性を確認する必要がある。

表 10 は、現在のキャッシュ・ディスクリプタおよび TLB ディスクリプタの値と、それに対応する特性を示している。このリストは、今後必要に応じて拡張される予定である。プロセッサのモデルとステッピングによっては、キャッシュ情報と TLB 情報のビット・フィールドの位置が変更される場合がある。したがって、ソフトウェアは、キャッシュ・ディスクリプタと TLB ディスクリプタを解析する際に、固定された位置を前提にしてはならない。

表 9. ディスクリプタのフォーマット

レジスタのビット 31	ディスクリプタのタイプ	説明
1	予約済み	将来に備えて予約。
0	8 ビット・ディスクリプタ	ディスクリプタは、キャッシュの特性を示すパラメータ・テーブルを指す。値が 0 の場合は、ディスクリプタは NULL である。

表 10. ディスクリプタのデコード値

値	キャッシュまたは TLB の説明
00h	NULL
01h	命令 TLB：4KB ページ、4 ウェイ・セット・アソシアティブ、32 エントリ
02h	命令 TLB：4MB ページ、フル・アソシアティブ、2 エントリ
03h	データ TLB：4KB ページ、4 ウェイ・セット・アソシアティブ、64 エントリ
04h	データ TLB：4MB ページ、4 ウェイ・セット・アソシアティブ、8 エントリ
06h	1 次命令キャッシュ：8KB、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
08h	1 次命令キャッシュ：16KB、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
0Ah	1 次データ・キャッシュ：8KB、2 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
0Ch	1 次データ・キャッシュ：16KB、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
22h	3 次キャッシュ：512 KB、4 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
23h	3 次キャッシュ：1MB、8 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
25h	3 次キャッシュ：2MB、8 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
29h	3 次キャッシュ：4MB、8 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
2Ch	1 次データ・キャッシュ：32KB、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
30h	1 次命令キャッシュ：32KB、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
39h	2 次キャッシュ：128KB、4 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
3Bh	2 次キャッシュ：128KB、2 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
3Ch	2 次キャッシュ：256KB、4 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
40h	2 次キャッシュなし。プロセッサが有効な 2 次キャッシュを搭載している場合は、3 次キャッシュなし。
41h	2 次キャッシュ：128KB、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
42h	2 次キャッシュ：256KB、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
43h	2 次キャッシュ：512KB、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
44h	2 次キャッシュ：1MB、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
45h	2 次キャッシュ：2MB、4 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
46h	3 次キャッシュ：4MB、4 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
47h	3 次キャッシュ：8MB、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
50h	命令 TLB：4KB、2MB または 4MB ページ、フル・アソシアティブ、64 エントリ

表 10. ディスクリプタのデコード値（続き）

値	キャッシュまたは TLB の説明
51h	命令 TLB：4KB、2MB または 4MB ページ、フル・アソシアティブ、128 エントリ
52h	命令 TLB：4KB、2MB または 4MB ページ、フル・アソシアティブ、256 エントリ
5Bh	データ TLB：4KB または 4MB ページ、フル・アソシアティブ、64 エントリ
5Ch	データ TLB：4KB または 4MB ページ、フル・アソシアティブ、128 エントリ
5Dh	データ TLB：4KB または 4MB ページ、フル・アソシアティブ、256 エントリ
60h	1 次データ・キャッシュ：16KB、8 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
66h	1 次データ・キャッシュ：8KB、4 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
67h	1 次データ・キャッシュ：16KB、4 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
68h	1 次データ・キャッシュ：32KB、4 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
70h	トレース・キャッシュ：12K uops、8 ウェイ・セット・アソシアティブ
71h	トレース・キャッシュ：16K uops、8 ウェイ・セット・アソシアティブ
72h	トレース・キャッシュ：32K uops、8 ウェイ・セット・アソシアティブ
78h	2 次キャッシュ：1MB、4 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
79h	2 次キャッシュ：128KB、8 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
7Ah	2 次キャッシュ：256KB、8 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
7Bh	2 次キャッシュ：512KB、8 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
7Ch	2 次キャッシュ：1MB、8 ウェイ・セット・アソシアティブ、セクタ・キャッシュ、64 バイト・ライン・サイズ
7Dh	2 次キャッシュ：2MB、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
7Fh	2 次キャッシュ：512KB、2 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
82h	2 次キャッシュ：256KB、8 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
83h	2 次キャッシュ：512KB、8 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
84h	2 次キャッシュ：1MB、8 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
85h	2 次キャッシュ：2MB、8 ウェイ・セット・アソシアティブ、32 バイト・ライン・サイズ
86h	2 次キャッシュ：512KB、4 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
87h	2 次キャッシュ：1MB、8 ウェイ・セット・アソシアティブ、64 バイト・ライン・サイズ
B0h	命令 TLB：4KB ページ、4 ウェイ・セット・アソシアティブ、128 エントリ
B3h	データ TLB：4KB ページ、4 ウェイ・セット・アソシアティブ、128 エントリ
F0h	64 バイト・プリフェッチ
F1h	128 バイト・プリフェッチ

3.7 Pentium® 4 プロセッサ、モデル 0 の出力例

Pentium 4 プロセッサ（モデル 0）は、表 8 に示した値を返す。AL の値=1 なので、レジスタの AL 以外のビットを解釈することは有効である。表 8 は、すべてのレジスタの MSB（ビット 31）が 0 であることを示している。これは、各レジスタに有効な 8 ビット・ディスクリプタが格納されていることを示す。表 8 のレジスタ値は、この Pentium 4 プロセッサが以下のキャッシュ特性と TLB 特性を持つことを示している。

- （66h）8 KB、4 ウェイ・セット・アソシアティブ、デュアル・セクタ・ライン（セクタサイズ 64 バイト）の 1 次データ・キャッシュ
- （5Bh）4KB または 4MB ページにマッピングされる、フル・アソシアティブ、64 エントリのデータ TLB
- （50h）4KB、2MB または 4MB ページにマッピングされる、フル・アソシアティブ、64 エントリの命令 TLB
- （7Ah）256KB、8 ウェイ・セット・アソシアティブ、デュアル・セクタ・ライン（セクタサイズ 64 バイト）の 2 次キャッシュ
- （70h）最大 12K の uops を格納できる、8 ウェイ・セット・アソシアティブのトレース・キャッシュ
- （40h）3 次キャッシュなし

表 11. 256KB L2 キャッシュを搭載した Pentium® 4 プロセッサ、モデル 0 の CPUID（EAX=2）戻り値の例

	31	23	15	7	0
EAX	66h	5Bh	50h	01h	
EBX	00h	00h	00h	00h	
ECX	00h	00h	00h	00h	
EDX	00h	7Ah	70h	40h	

4 プロセッサ・シリアル・ナンバ

プロセッサ・シリアル・ナンバは、プロセッサ識別の概念を拡張する機能である。プロセッサ・シリアル・ナンバは、CPUID 命令によってアクセスできる 96 ビットの数である。アプリケーションは、プロセッサ・シリアル・ナンバを使用して、プロセッサ（また、拡張機能により、プロセッサが搭載されているシステム）を識別できる。

プロセッサ・シリアル・ナンバは、プロセッサごとに、ソフトウェアによってアクセスできる識別子を提供する。プロセッサ・シリアル・ナンバと他の指示子を組み合わせて、ユーザの識別に応用することもできる。プロセッサ・シリアル・ナンバの用途には、メンバー認証、データのバックアップ / 復元の保護、リムーバブル記憶装置のデータ保護、ファイルアクセスの管理、適切なユーザ間の文書交換の保証などが含まれる。

プロセッサ・シリアル・ナンバは、資産管理、製品トラッキング、リモートシステムのロードとコンフィグレーション、ブートアップ・コンフィグレーションの支援に使用できるツールとなる。システムサービスの場合、プロセッサ・シリアル・ナンバを使用して、ヘルプデスクにアクセスしているユーザの識別や、エラーレポートのトラッキングが可能になる。プロセッサ・シリアル・ナンバはプロセッサの識別子を提供するが、常に独自の値であるとは限らない。プロセッサの動作モードによっては、間違ったプロセッサ・シリアル・ナンバが返される可能性がある。例えば、プロセッサが推奨動作仕様（電圧、動作周波数など）の範囲外で動作している場合、プロセッサからプロセッサ・シリアル・ナンバを正しく読み取れないことがある。また、BIOS またはソフトウェアの動作が不適当な場合、不正確なプロセッサ・シリアル・ナンバが生成されることがある。こうした条件下では、間違ったプロセッサ・シリアル・ナンバや重複するプロセッサ・シリアル・ナンバが返される可能性がある。システムメーカは、システムに冗長機能やフォールト・トレランス手法を組み込むことにより、プロセッサ・シリアル・ナンバ機能の安定性を強化できる。

他の独立した数値に対する選択条件としてプロセッサ・シリアル・ナンバを使用し、電気的にアクセス可能な数値を生成すれば、独自の値になりそうである。プロセッサ・シリアル・ナンバは、信頼できるコネクテッド PC を可能にするための便利なビルディング・ブロックの 1 つである。

4.1 プロセッサ・シリアル・ナンバの有無

プロセッサ・シリアル・ナンバ機能のサポートを確認するには、プログラムは、次のように、EAX レジスタのパラメータ値を "1" に設定し、CPUID 命令を実行する。

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

CPUID 命令の実行後、ECX レジスタと EDX レジスタに機能フラグが返される。PSN 機能フラグ（EDX レジスタ、ビット 18）の値が "1" の場合は、プロセッサ・シリアル・ナンバ機能がサポートされ、イネーブルになっている。**PSN 機能フラグの値が "0" の場合は、プロセッサ・シリアル・ナンバ機能がサポートされていないか、または Pentium® III プロセッサ上でディスエーブルになっている。**

4.2 96 ビットのプロセッサ・シリアル・ナンバの構成

96ビットのプロセッサ･シリアル･ナンバは､3つの32ビット･エンティティを連結したものである。

プロセッサ・シリアル・ナンバの最上位32ビットにアクセスするには、プログラムは、次のように、EAXレジスタのパラメータ値を"1"に設定し、CPUID命令を実行する。

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

CPUID命令の実行後、EAXレジスタにはプロセッサ・シグニチャが返される。プロセッサ・シグニチャは、プロセッサ・シリアル・ナンバの最上位32ビットを構成する。プロセッサ・シリアル・ナンバの下位64ビットを収集する前に、EAX内の値を保存する必要がある。

プロセッサ・シリアル・ナンバの下位64ビットにアクセスするには、プログラムは、次のように、EAXレジスタのパラメータ値を"3"に設定し、CPUID命令を実行する。

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

CPUID命令の実行後、EDXレジスタにはプロセッサ・シリアル・ナンバの中間の32ビットが返され、ECXレジスタには最下位32ビットが返される。ソフトウェアは、保存したプロセッサ・シグニチャ、EDX、ECXを連結し、96ビットのプロセッサ・シリアル・ナンバ全体を返す。

プロセッサ・シリアル・ナンバは、4ケタの16進数を6つ並べて表示される（例：XXXX-XXXX-XXXX-XXXX-XXXX-XXXX、Xは16進数の1ケタを表す）。16進数の英字は大文字で表示される。

5 ブランド ID とブランド・ストリング

5.1 ブランド ID

Pentium® III プロセッサ（モデル 8）、Pentium® III Xeon™ プロセッサ（モデル 8）、Celeron® プロセッサ（モデル 8）から、ブランド ID 機能が追加され、プロセッサ識別の概念がさらに拡張された。ブランド ID は、CPUID によってアクセスできる 8 ビットの数である。アプリケーションは、プロセッサの識別にブランド ID を利用できる。

ブランド ID 機能を実装しているプロセッサは、EAX=1 に設定して CPUID 命令を実行した場合、EBX レジスタのビット 7 ～ 0 にブランド ID を返す（表 8 を参照）。ブランド ID 機能をサポートしていないプロセッサは、EBX レジスタのビット 7 ～ 0 に 0 を返す。

Pentium® II プロセッサ、Pentium® II Xeon プロセッサ、Celeron プロセッサ、Pentium III プロセッサ、Pentium III Xeon プロセッサの以前のモデルを区別する場合、アプリケーション・ソフトウェアは、L2 キャッシュ・ディスクリプタに依存していた。しかし、この方法では、結果があいまいになる場合があった。例えば、ソフトウェアは、Pentium II プロセッサと 512KB L2 キャッシュを搭載した Pentium II Xeon プロセッサを厳密に区別できなかった。ブランド ID は、プロセッサ・ブランドごとに、ソフトウェアによってアクセスできる独自の値を提供し、このようなあいまいさを解消する。表 12 は、各プロセッサについて定義されているブランド ID の値を示している。

5.2 ブランド・ストリング

ブランド・ストリングは、Pentium® 4 プロセッサを含む数種類のインテル® IA32 プロセッサに実装されている、CPUID 命令の新しい拡張機能である。今後の IA32 アーキテクチャ・ベースのプロセッサは、ブランド・ストリング機能を使用して、拡張された CPUID 命令に対して、プロセッサの ASCII ブランド識別ストリングと最大動作周波数を返す。なお、返される周波数は、プロセッサの検証済みの最大動作周波数であり、プロセッサの現在の動作周波数ではない。

EAX を表 1 の値に設定して CPUID を実行した場合、プロセッサは、表 1 の説明に従って汎用レジスタに ASCII ブランド・ストリングを返す。

ブランド / 周波数ストリングの長さは 48 文字として定義され、47 バイトは文字であり、48 バイト目は NULL（0）になる。EAX = 80000002h、80000003h、80000004h に設定して CPUID を実行した場合、ストリングが NULL で終わっており、プロセッサが有効なデータを返しさえすれば、プロセッサは 47 文字未満の ASCII ストリングを返すことがある。

cpuid3a.asm プログラムは、ソフトウェアがブランド・ストリングを作成する方法を示している（例 1 を参照）。プロセッサがブランド・ストリングをサポートしているかを確認するには、ソフトウェアは以下の手順に従う必要がある。

1. EAX=80000000h に設定して CPUID 命令を実行する。
2. （（EAX 内の戻り値）> 80000000h）の場合は、プロセッサは拡張された CPUID 機能をサポートしており、EAX にはサポートされる拡張機能のうち最大の値が返される。
3. EAX >= 80000004h の場合は、プロセッサ・ブランド・ストリング機能がサポートされている。

表 12. ブランド ID、CPUID（EAX=1）実行後の EBX の戻り値（ビット 7 ～ 0）

値	説明
00h	サポートしていない
01h	インテル® Celeron® プロセッサ
02h	インテル® Pentium® III プロセッサ
03h	インテル® Pentium® III Xeon™ プロセッサ プロセッサ・シグニチャ =000006B1h の場合は、インテル® Celeron® プロセッサ
04h	インテル® Pentium® III プロセッサ
06h	モバイル インテル® Pentium® III プロセッサ - M
07h	モバイル インテル® Celeron® プロセッサ
08h	インテル® Pentium® 4 プロセッサ プロセッサ・シグニチャ >=00000F13h の場合は、インテル® 純正プロセッサ
09h	インテル® Pentium® 4 プロセッサ
0Ah	インテル® Celeron® プロセッサ
0Bh	インテル® Xeon™ プロセッサ プロセッサ・シグニチャ <00000F13h の場合は、インテル® Xeon™ プロセッサ MP
0Ch	インテル® Xeon™ プロセッサ MP
0Eh	モバイル インテル® Pentium® 4 プロセッサ - M プロセッサ・シグニチャ <00000F13h の場合は、インテル® Xeon™ プロセッサ
0Fh	モバイル インテル® Celeron® プロセッサ
11h	モバイル インテル® 純正プロセッサ
12h	インテル® Celeron® M プロセッサ
13h	モバイル インテル® Celeron® プロセッサ
14h	インテル® Celeron® プロセッサ
15h	モバイル インテル® 純正プロセッサ
16h	インテル® Pentium® M プロセッサ
17h	モバイル インテル® Celeron® プロセッサ
その他のすべての値	予約済み

表 13. プロセッサ・ブランド・ストリング機能

EAX の入力値	機能	戻り値
80000000h	サポートしている拡張機能のうち最大の値	EAX= サポートしている拡張機能のうち最大の値、 EBX = ECX = EDX = 予約済み
80000001h	拡張プロセッサ・シグニチャと拡張機能ビット	EDX と ECX に拡張機能フラグが返される。 EAX = EBX = 予約済み
80000002h	プロセッサ・ブランド・ストリング	EAX、EBX、ECX、EDX に ASCII ブランド・ストリングが返される
80000003h	プロセッサ・ブランド・ストリング	EAX、EBX、ECX、EDX に ASCII ブランド・ストリングが返される
80000004h	プロセッサ・ブランド・ストリング	EAX、EBX、ECX、EDX に ASCII ブランド・ストリングが返される

6 使用ガイドライン

本書は、インテルが推奨する機能検出手法について説明している。ソフトウェアは、非正規のプログラミング手法、未公開の機能、または本書で説明するガイドラインとは異なる方法を利用して、プロセッサの機能を特定しようとしてはならない。

以下のガイドラインは、プログラマが開発するソフトウェアについて、極めて幅広い互換性を維持できるようにすることを目的にしている。

- CPUID 命令を検出する場合、CPUID オペコードの実行時に無効オペコード・トラップが発生しないことを判定の基準にしてはならない。32 ビット・プロセッサを検出する場合、PUSHFD オペコードの実行時に無効オペコード・トラップが発生しないことを判定の基準にしてはならない。第 2 章と第 7 章の説明に従って、ID フラグをテストすること。
- **特定のファミリまたはモデルが常に特定の機能を持つと考えてはならない。例えば、ファミリ値が 5 のプロセッサ（Pentium® プロセッサ）が常にオンチップの浮動小数点ユニットを持つと考えてはならない。この判定には機能フラグを使用すること。**
- 上位のファミリまたはモデル番号のプロセッサが、下位のファミリまたはモデル番号のプロセッサのすべての機能を持つと考えてはならない。例えば、ファミリ値が 6 のプロセッサ（P6 ファミリ・プロセッサ）は、必ずしもファミリ値が 5 のプロセッサのすべての機能を持つわけではない。
- OverDrive® プロセッサの機能と、それに対応する OEM 版プロセッサの機能が同一であると考えてはならない。内蔵キャッシュや命令の実行方法が異なる場合がある。
- ステッピングまたは機能を特定する場合、プロセッサの未公開の機能を利用してはならない。例えば、Intel386™ プロセッサ A ステップのビット命令の一部は、B ステップでは廃止されている。そこで、一部のソフトウェアはこれらの命令の実行を試み、無効オペコード例外が発生した場合、A ステップのプロセッサではないと判定していた。ところが、Intel486™ プロセッサが同じオペコードを異なる命令に使用したため、このソフトウェアは正常に動作しなくなった。この場合は、プロセッサ・シグニチャ内のステッピング情報を使用するのが正しい方法である。
- 機能フラグは個別にテストすること。未定義のビットを設計の前提にしてはならない。例えば、比較命令を使用して機能レジスタと 2 進数の 1 を比較する方法で FPU ビットをテストすると、判定に失敗することがある。
- 特定のファミリまたはモデルのクロックが常に特定の周波数で動作すると考えてはならない。また、タイミング・ループなど、プロセッサの動作周波数に依存するコードを作成してはならない。例えば、OverDrive プロセッサは、高い内部クロック周波数で動作するにもかかわらず、同じファミリまたはモデルを報告することがある。プロセッサ・コアのクロック周波数を測定する場合は、プロセッサ・コアを直接測定できるように、システムのタイマを組み合わせて経過時間と TSC（タイムスタンプ・カウンタ）を測定すること。詳細については、第 11 章と例 6 を参照のこと。

- プロセッサ・モデル固有レジスタは、プロセッサ間で異なる場合がある（Pentium プロセッサの各種モデルを含む）。モデル固有レジスタは、システムに組み込まれているプロセッサのモデル固有レジスタの内容が確認済みの場合にのみ使用すること。OverDrive プロセッサによってアップグレード可能なシステムでは、このことが特に重要である。当該プロセッサの BIOS 開発者ガイドに定義されているモデル固有レジスタだけを使用すること。
- 仮想 8086 モードで CPUID 命令を実行する場合は、CPUID アルゴリズムの結果を判定の基準にしてはならない。
- モデル番号またはステッピング番号の順序を前提にしてはならない。これらの番号は任意に割り当てられる。
- その他の選択条件と組み合わされない限り、プロセッサ・シリアル・ナンバが常に独自の番号であると考えてはならない。
- プロセッサ・シリアル・ナンバは、4 ケタの 16 進数を 6 つ並べて表示する（例 : XXXX-XXXX-XXXX-XXXX-XXXX-XXXX、X は 16 進数の 1 ケタを表す）。
- 16 進数の英字は大文字で表示する。
- プロセッサ・シリアル・ナンバの下位 64 ビットが 0 の場合は、プロセッサ上でプロセッサ・シリアル・ナンバが無効であるか、サポートされていないか、ディスエーブルにされていることを示す。

7 適切な識別シーケンス

CPUID 命令を使用してプロセッサを識別するには、ソフトウェアは以下の手順に従う。

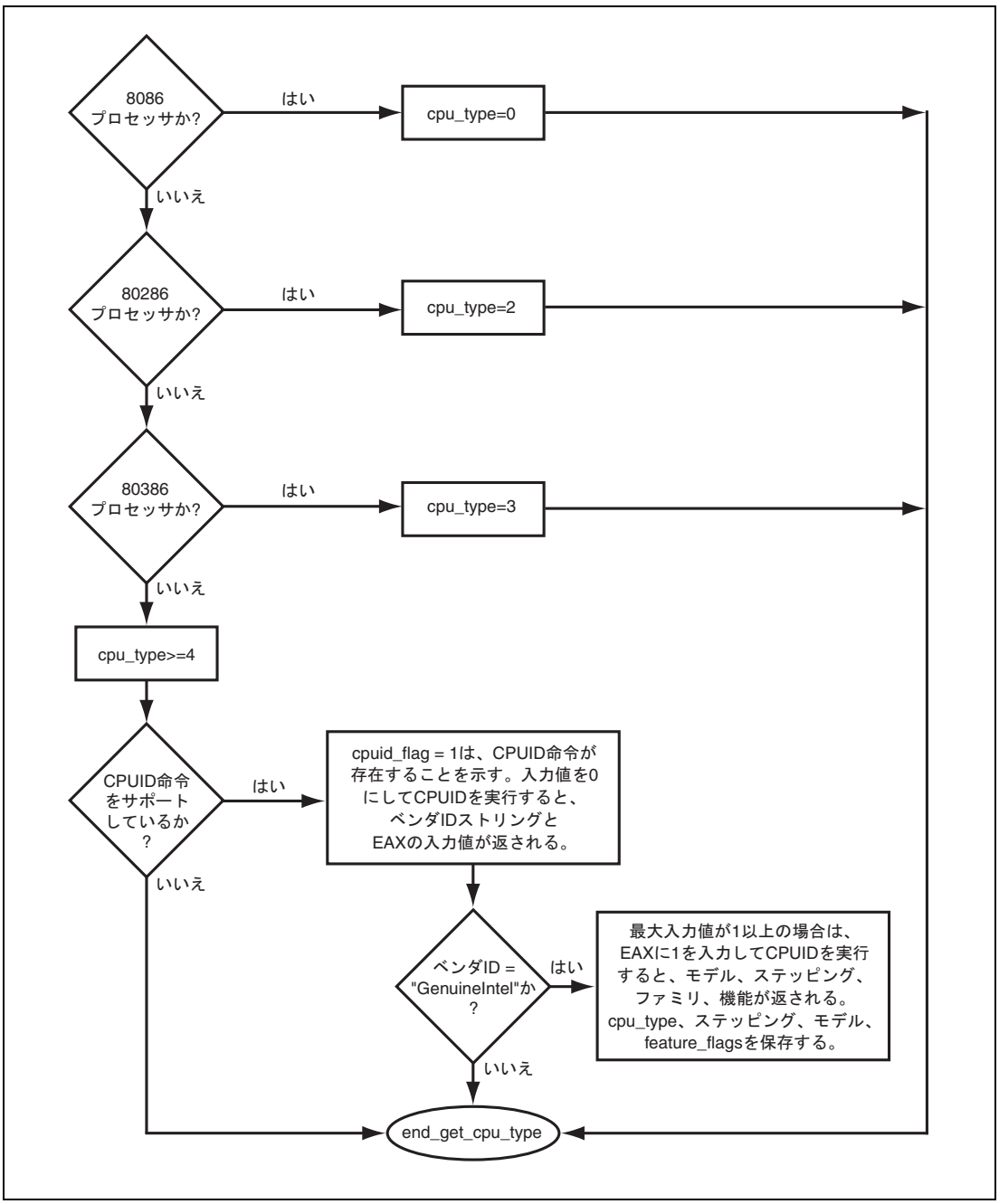
1. EFLAGS レジスタ内の ID フラグを変更することにより、CPUID 命令のサポートを確認する。ID フラグを変更できない場合は、CPUID 命令を使用してプロセッサを識別することはできない。
2. EAX の値を 80000000h に設定して CPUID 命令を実行する。CPUID 機能 80000000h を使用して、ブランド・ストリングがサポートされているかを確認できる。CPUID 機能 80000000h が 80000004h またはそれより大きい値を EAX に返した場合は、ブランド・ストリング機能がサポートされているので、ソフトウェアは CPUID 機能 80000002h ～ 80000004h を使用してプロセッサを識別できる。
3. ブランド・ストリング機能がサポートされていない場合は、EAX = 1 に設定して CPUID を実行する。CPUID 機能 1 は、EAX レジスタにプロセッサ・シグニチャを返し、EBX レジスタのビット 0 ～ 7 にブランド ID を返す。EBX レジスタのビット 0 ～ 7 に 0 でない値が返された場合は、ブランド ID がサポートされている。ソフトウェアは、ブランド ID のリスト（表 1 を参照）をスキャンしてプロセッサを識別できる。
4. ブランド ID 機能がサポートされていない場合は、ソフトウェアは、プロセッサ・シグニチャ（図 2 を参照）とキャッシュ・ディスクリプタ（表 10 を参照）を組み合わせて、プロセッサを識別する必要がある。

cpuid3a.asm プログラム例は、CPUID 命令の正しい使用法を示している（例 1 を参照）。このプログラム例は、ブランド・ストリング、ブランド ID、プロセッサ・シグニチャ、CPUID 命令を実装していない古い世代のプロセッサ（図 5 を参照）を識別する方法も示している。このプログラム例には、以下の 2 のプロシージャが含まれている。

- `get_cpu_type` は、プロセッサ・タイプを特定する。図 5 はこのプロシージャのフローを示している。
- `get_fpu_type` は、浮動小数点ユニット（FPU）または数値演算コプロセッサ（MCP）のタイプを判定する。

このプロシージャは、8086 プロセッサ、80286 プロセッサ、Intel386™ プロセッサ、Intel486™ プロセッサ、Pentium® プロセッサ、MMX® テクノロジ Pentium® プロセッサ、MMX® テクノロジ OverDrive® プロセッサ、Pentium® Pro プロセッサ、Pentium® II プロセッサ、Pentium® II Xeon™ プロセッサ、Pentium® II OverDrive® プロセッサ、Celeron® プロセッサ、Pentium® III プロセッサ、Pentium® III Xeon™ プロセッサ、Pentium® 4 プロセッサでテスト済みである。このプログラム例はアセンブリ言語で記述され、ランタイム・ライブラリに組み込んだり、オペレーティング・システム内にシステムコールとして組み込むのに適している。

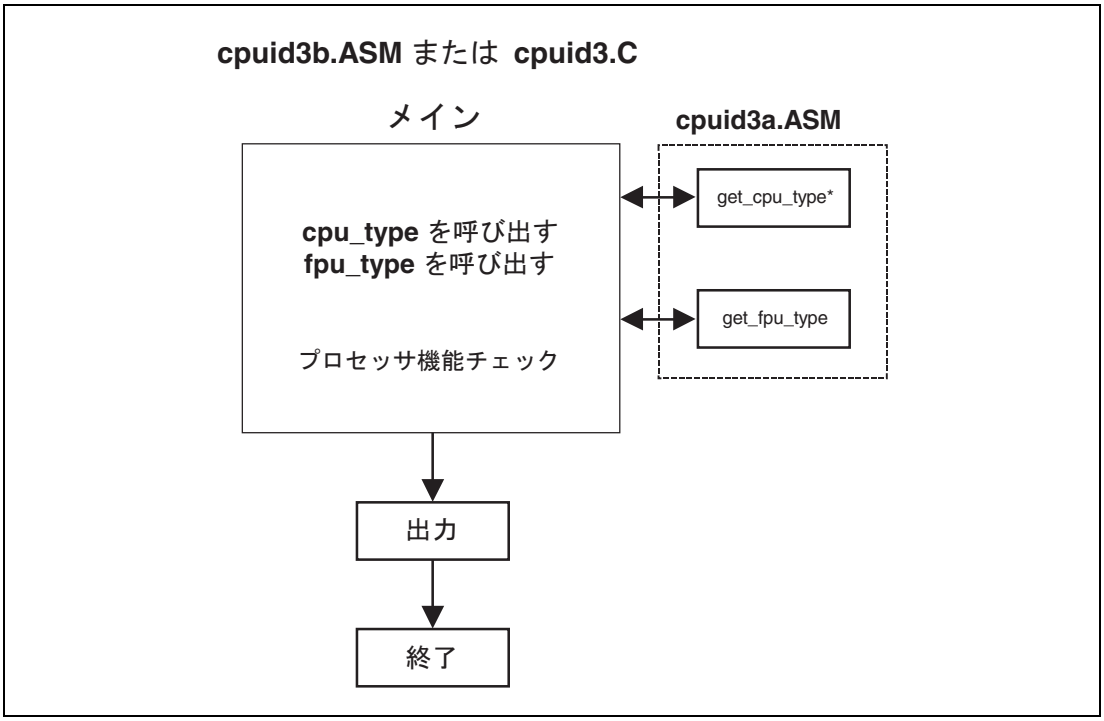
図 5. プロセッサの get_cpu_type プロシージャのフロー



8 プログラム例の使用法

cpuid3b.asm と cpuid3.c プログラム例は、get_cpu_type と get_fpu_type プロシージャを呼び出し、返された情報を解釈するアプリケーションを示している。例 2 と例 3 にこのコードを示す。モニタに表示される結果から、組み込まれているプロセッサとその機能を特定できる。cpuid3b.asm の例はアセンブリ言語で記述され、返された情報を DOS 環境内で表示するアプリケーションを示している。cpuid3.c の例は C 言語で記述されている（例 2 と例 3 を参照）。図 6 は、cpuid3a.asm、cpuid3b.asm、cpuid3.c の 3 つのプログラム例の関係を簡単に示している。

図 6. プロセッサ識別機能抽出プロシージャのフロー



9 機能検出の代替手法

一部の機能フラグは、拡張命令セット（すなわち、MMX® テクノロジ、SSE、SSE2）のサポートを表示する。拡張命令のサポートを判定する際の推奨手順は、CPUID 命令を使用して、機能フラグをテストする方法である。ただし、拡張命令のサポートを判定する代替手段として、例外ハンドラをインストールし、拡張命令を 1 つ実行する方法もある。例外が発生せずに命令が実行されれば、プロセッサはその拡張命令セットをサポートしている。例外が発生して例外ハンドラが実行された場合は、プロセッサはその拡張命令をサポートしていない。例外ハンドラをインストールする前に、ソフトウェアは、EAX = 0 に設定して CPUID 命令を実行する必要がある。この CPUID 命令でインテル・ベンダ ID ストリング "GenuineIntel" が返されれば、ソフトウェアはインテルの拡張命令に関するテストを実行できる。CPUID 命令がインテル・ベンダ ID を返しさえすれば、将来のインテル・プロセッサでもこの方法を使用できる。この方法では、ソフトウェアはファミリとモデルをチェックする必要はない。

features.cpp プログラムは、C++ 言語で記述されている（例 4 を参照）。このプログラムは、例外を利用して SSE3、SSE2、SSE、MMX テクノロジの各拡張命令のサポートを判定する方法を示している。features.cpp は、以下の手順を実行する。

1. vendor-ID == "GenuineIntel" であることを確認する
2. SSE3 のテスト用の例外ハンドラをインストールする
3. SSE3 の実行を試みる（haddpd xmm1, xmm2）
4. SSE2 のテスト用の例外ハンドラをインストールする
5. SSE2 の実行を試みる（paddq xmm1, xmm2）
6. SSE のテスト用の例外ハンドラをインストールする
7. SSE の実行を試みる（orps xmm1, xmm2）
8. MMX テクノロジのテスト用の例外ハンドラをインストールする
9. MMX 命令の実行を試みる（emms）
10. サポートしている拡張命令セットを出力する

10 デノーマル・ゼロ

SSE2 の導入とともに、一部のインテル® アーキテクチャ・プロセッサには、SSE と SSE2 のソース・オペランドのデノーマル数を 0 に変換する機能が追加された。この機能は、デノーマル・ゼロ（DAZ）と呼ばれている。DAZ モードは、IEEE 規格 754 に適合しない。DAZ モードは、ストリーミング・メディア処理などのアプリケーションにおけるプロセッサ・パフォーマンスの向上を目的としている。デノーマル・オペランドを 0 に丸めると、処理後のデータの品質が多少低下する。

一部のプロセッサ・ステッピングは、SSE2 をサポートしているが、DAZ モードをサポートしていない。プロセッサが DAZ モードをサポートしているかを確認するには、ソフトウェアは以下の手順を実行する必要がある。

1. 入力値を EAX=0 にして CPUID 命令を実行し、ベンダ ID ストリング "GenuineIntel" が返されることを確認する。
2. EAX=1 に設定して CPUID 命令を実行する。EDX レジスタに機能フラグがロードされる。
3. FXSR 機能フラグ（EDX ビット 24）がセットされていることを確認する。これは、プロセッサが FXSAVE 命令と FXRSTOR 命令をサポートしていることを示す。
4. XMM 機能フラグ（EDX ビット 25）または EMM 機能フラグ（EDX ビット 26）がセットされていることを確認する。これは、プロセッサが、SSE と SSE2 のうち少なくとも一方と、その命令セットの MXCSR 制御レジスタをサポートしていることを示す。
5. 16 バイトにアライメントされた 512 バイトのメモリ領域をゼロにする。FXSAVE の一部の実装は、FXSAVE イメージ内の予約済み領域を変更しないため、この操作が必要になる。
6. クリアされた領域への FXSAVE を実行する。
7. FXSAVE イメージのバイト 28 ～ 31 は、MXCSR_MASK を格納するように定義されている。この値が 0 の場合、プロセッサの MXCSR_MASK は 0xFFBF である。0 でない場合、MXCSR_MASK はこのダブルワードの値である。
8. MXCSR_MASK のビット 6 がセットされている場合は、DAZ がサポートされている。

このアルゴリズムの完了後、DAZ がサポートされている場合、ソフトウェアは、MXCSR レジスタの保存領域のビット 6 をセットして FXRSTOR 命令を実行し、DAZ モードをイネーブルにできる。あるいは、ソフトウェアは、LDMXCSR 命令を実行して MXCSR のビット 6 をセットし、DAZ モードをイネーブルにできる。『インテル® アーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、上巻：基本アーキテクチャ』の「ストリーミング SIMD 拡張命令（SSE）によるプログラミング」の章を参照のこと。

アセンブリ言語プログラム dazdtect.asm（例 5 を参照）は、この DAZ 検出アルゴリズムを示している。

11 動作周波数

タイムスタンプ・カウンタの導入とともに、リング 0 特権でリアルモードまたは保護モードで動作しているソフトウェアが、プロセッサの実際の動作周波数を計算できるようになった。ソフトウェアが動作周波数を計算するには、基準期間が必要である。基準期間は、周期的な割り込みか、(システムクロックではなく）時間に基づく他のタイマである。ソフトウェアは、基準期間の始まりと終わりにタイムスタンプ・カウンタを読み取る必要がある。ソフトウェアは、RDTSC 命令を実行するか、または ECX レジスタを 10h に設定して RDMSR 命令を実行することによって、TSC を読み取ることができる。いずれの命令も、現在の 64 ビット TSC を EDX：EAX レジスタペアにコピーする。

プロセッサの動作周波数を確認するには、ソフトウェアは以下の手順を実行する。アセンブリ言語プログラム frequenc.asm（例 6 を参照）は、周波数検出アルゴリズムを示している。

1. 入力値を EAX = 0 にして CPUID 命令を実行し、ベンダ ID ストリング "GenuineIntel" が返されることを確認する。
2. EAX=1 に設定して CPUID 命令を実行し、EDX レジスタに機能フラグをロードする。
3. TSC 機能フラグ（EDX ビット 4）がセットされていることを確認する。これは、プロセッサがタイムスタンプ・カウンタと RDTSC 命令をサポートしていることを示す。
4. 基準期間の始まりで TSC を読み取る。
5. 基準期間の終わりで TSC を読み取る。
6. 基準期間の始まりと終わりから TSC の差分を計算する。
7. TSC の差分を基準期間で割り、実際の周波数を計算する。

実際の周波数 =（終わりの TSC 値 - 始まりの TSC 値）/ 基準期間

注意：測定値の精度は、基準期間の精度によって異なる。基準期間が長いほど、正確な結果が得られる。また、計算を複数回繰り返すと精度が向上する。

例 1. プロセッサ識別機能抽出プロシージャ

```
;       Filename:  cpuid3a.asm
;       Copyright (c) Intel Corporation 1993-2004
;
;       This program has been developed by Intel Corporation.  Intel
;       has various intellectual property rights which it may assert
;       under certain circumstances, such as if another
;       manufacturer's processor mis-identifies itself as being
;       "GenuineIntel" when the CPUID instruction is executed.
;
;       Intel specifically disclaims all warranties, express or
;       implied, and all liability, including consequential and other
;       indirect damages, for the use of this program, including
;       liability for infringement of any proprietary rights,
;       and including the warranties of merchantability and fitness
;       for a particular purpose.  Intel does not assume any
;       responsibility for any errors which may appear in this program
;       nor any responsibility to update it.
;
;       This code contains two procedures:
;       _get_cpu_type: Identifies processor type in _cpu_type:
;               0=8086/8088 processor
;               2=Intel 286 processor
;               3=Intel386(TM) family processor
;               4=Intel486(TM) family processor
;               5=Pentium(R) family processor
;               6=P6 family of processors
;               F=Pentium 4 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(R) processor, Pentium(R) Pro
;       processor, Pentium(R) II processor, Pentium II Xeon(TM) processor,
;       Pentium II Overdrive(R), Intel Celeron processor, Pentium III processor,
;       Pentium III Xeon processor, Pentium 4 processors and
;       Intel(R) Xeon(TM) processors.

;       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
;
;       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_ebx
        public  _features_ecx
        public  _features_edx
        public  _ext_funct_1_eax
        public  _ext_funct_1_ebx
        public  _ext_funct_1_ecx
        public  _ext_funct_1_edx
        public  _ext_funct_6_eax
        public  _ext_funct_6_ebx
        public  _ext_funct_6_ecx
        public  _ext_funct_6_edx
        public  _ext_funct_8_eax
        public  _ext_funct_8_ebx
        public  _ext_funct_8_ecx
        public  _ext_funct_8_edx
        public  _cache_eax
        public  _cache_ebx
        public  _cache_ecx
        public  _cache_edx
        public  _sep_flag
        public  _brand_string

        _cpu_type       db      0
        _fpu_type       db      0
        _v86_flag       db      0
        _cpuid_flag     db      0
        _intel_CPU      db      0
        _sep_flag       db      0
        _vendor_id      db      "------------"
        intel_id        db      "GenuineIntel"
        _cpu_signature  dd      0
        _features_ebx   dd      0
        _features_ecx   dd      0
        _features_edx   dd      0
        _ext_funct_1_eax                dd      0
        _ext_funct_1_ebx        dd      0
        _ext_funct_1_ecx                dd      0
        _ext_funct_1_edx        dd      0
        _ext_funct_6_eax                dd      0
        _ext_funct_6_ebx        dd      0
        _ext_funct_6_ecx                dd      0
        _ext_funct_6_edx        dd      0
```

```
        _ext_funct_8_eax            dd      0
        _ext_funct_8_ebx        dd      0
        _ext_funct_8_ecx            dd      0
        _ext_funct_8_edx        dd      0
        _cache_eax      dd      0
        _cache_ebx      dd      0
        _cache_ecx      dd      0
        _cache_edx      dd      0
        fp_status       dw      0
        _brand_string   db      48 dup (0)

.code
;
;       comment this line for 32-bit segments
;
.8086
;
;       uncomment this line for 32-bit segments
;
;       .386

;*************************************************************************
        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.

.286
check_80286:
        smsw    ax              ; save machine status word
        and     ax, 1           ; isolate PE bit of MSW
        mov     _v86_flag, al   ; save PE bit to indicate V86

        or      cx, 0f000h      ; try to set bits 12-15
        push    cx              ; save new FLAGS value on stack
        popf                    ; replace current FLAGS value
        pushf                   ; get new FLAGS
        pop     ax              ; store new FLAGS in AX
        and     ax, 0f000h      ; if bits 12-15 are clear
        mov     _cpu_type, 2    ; processor=80286, turn on 80286 flag
        jz      end_cpu_type    ; jump if processor is 80286

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

.386
;
;       "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    ; jump if 80386 processor
        push    ecx
        popfd                   ; restore AC bit in EFLAGS first
;       Intel486 processor check
;       Checking for ability to set/clear ID flag (Bit 21) in EFLAGS
;       which indicates the presence of a processor with the CPUID
;       instruction.

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

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

        mov     _cpuid_flag, 1          ; flag indicating use of CPUID inst.
        push    ebx                     ; save registers
        push    esi
        push    edi

        mov     eax, 0                  ; set up for CPUID instruction
        CPU_ID                          ; get and save vendor ID

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

        cmp     dword ptr intel_id, ebx
        jne     end_cpuid_type
        cmp     dword ptr intel_id[+4], edx
        jne     end_cpuid_type
        cmp     dword ptr intel_id[+8], ecx
        jne     end_cpuid_type          ; if not equal, not an Intel processor

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

        mov     eax, 1
        CPU_ID                          ; 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
        cmp     ax, 2                   ; Are cache descriptors supported?
        jl      end_cpuid_type

        mov     eax, 2                  ; set up to read cache descriptor
        CPU_ID
        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
                                        ; information
```

```
        mov     eax, 80000000h          ; check if brand string is supported
        CPU_ID
        cmp     eax, 80000004h
        jbe     end_cpuid_type          ; take jump if not supported

        mov     eax, 80000001h          ; Get the Extended Feature Flags
        CPU_ID

        mov     _ext_funct_1_eax, eax
        mov     _ext_funct_1_ebx, ebx
        mov     _ext_funct_1_ecx, ecx
        mov     _ext_funct_1_edx, edx

        mov     di, offset _brand_string

        mov     eax, 80000002h          ; get first 16 bytes of brand string
        CPU_ID
        mov     dword ptr [di], eax     ; save bytes 0 .. 15
        mov     dword ptr [di+4], ebx
        mov     dword ptr [di+8], ecx
        mov     dword ptr [di+12], edx
        add     di, 16

        mov     eax, 80000003h
        CPU_ID
        mov     dword ptr [di], eax     ; save bytes 16 .. 31
        mov     dword ptr [di+4], ebx
        mov     dword ptr [di+8], ecx
        mov     dword ptr [di+12], edx
        add     di, 16

        mov     eax, 80000004h
        CPU_ID
        mov     dword ptr [di], eax     ; save bytes 32 .. 47
        mov     dword ptr [di+4], ebx

        mov     dword ptr [di+8], ecx
        mov     dword ptr [di+12], edx

        mov     eax, 80000000h          ; check if L2 Cache Features supported
        CPU_ID
        cmp     eax, 80000006h
        jbe     end_cpuid_type          ; take jump if not supported

        mov     _ext_funct_6_eax, eax
        mov     _ext_funct_6_ebx, ebx
        mov     _ext_funct_6_ecx, ecx
        mov     _ext_funct_6_edx, edx

        mov     eax, 80000000h          ; check if Address Size function supported
        CPU_ID
        cmp     eax, 80000008h
        jbe     end_cpuid_type          ; take jump if not supported

        mov     _ext_funct_8_eax, eax
        mov     _ext_funct_8_ebx, ebx
        mov     _ext_funct_8_ecx, ecx
        mov     _ext_funct_8_edx, edx

end_cpuid_type:
```

```
        pop     edi                     ; restore registers
        pop     esi
        pop     ebx
;
;       comment this line for 32-bit segments
;
.8086
end_cpu_type:
        ret
_get_cpu_type   endp

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

        public  _get_fpu_type
        _get_fpu_type   proc

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

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

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

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

;       80287/80387 check for the Intel386 processor

check_infinity:
        cmp     _cpu_type, 3
        jne     end_fpu_type
        fld1                            ; must use default control from FNINIT
        fldz                            ; form infinity
        fdiv                            ; 8087/Intel287 NDP say +inf = -inf
        fld     st                      ; form negative infinity
        fchs                            ; Intel387 NDP says +inf <> -inf
        fcompp                          ; see if they are the same
        fstsw   fp_status               ; look at status from FCOMPP
        mov     ax, fp_status
```

```
        mov     _fpu_type, 2        ; store Intel287 NDP for FPU type
        sahf                        ; see if infinities matched
        jz      end_fpu_type        ; jump if 8087 or Intel287 is present
        mov     _fpu_type, 3        ; store Intel387 NDP for FPU type
end_fpu_type:
        ret
_get_fpu_type   endp

        end
```

例 2. アセンブリ言語で記述されたプロセッサ識別プロシージャ

```
;     Filename:  cpuid3b.asm
;          Copyright (c) Intel Corporation 1993-2004
;
;          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(R), Pentium(R) Pro,
;          Pentium(R) II processors, Pentium(R) II Xeon(TM) processors, Pentium(R) II
;          Overdrive(R) processors,  Intel(R) Celeron(R) processors, Pentium(R) III
;          processors, Pentium(R) III Xeon(TM) processors, Pentium(R) 4 processors
;          and Intel(R) Xeon(TM) processors DP and MP when executed in the
;          real-address mode.

;   NOTE: This code is written using 16-bit Segments

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

           TITLE    cpuid3b

DOSSEG
.model   small

.stack    100h

OP_O     MACRO
           db       66h                      ; hardcoded operand override
ENDM
.data
           extrn    _cpu_type:      byte
           extrn    _fpu_type:      byte
           extrn    _cpuid_flag:    byte
           extrn    _intel_CPU:     byte
           extrn    _vendor_id:     byte
```

```
        extrn   _cpu_signature: dword
        extrn   _features_ecx:  dword
        extrn   _features_edx:  dword
        extrn   _features_ebx:  dword
        extrn   _ext_funct_1_eaxdword
        extrn   _ext_funct_1_ebxdword
        extrn   _ext_funct_1_ecxdword
        extrn   _ext_funct_1_edxdword
        extrn   _ext_funct_6_eaxdword
        extrn   _ext_funct_6_ebxdword
        extrn   _ext_funct_6_ecxdword
        extrn   _ext_funct_6_edxdword
        extrn   _ext_funct_8_eaxdword
        extrn   _ext_funct_8_ebxdword
        extrn   _ext_funct_8_ecxdword
        extrn   _ext_funct_8_edxdword
        extrn   _cache_eax:     dword
        extrn   _cache_ebx:     dword
        extrn   _cache_ecx:     dword
        extrn   _cache_edx:     dword
        extrn   _brand_string:  byte

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

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

        mov     ax, 4c00h
        int     21h

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

        extrn   _get_cpu_type: proc

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

        extrn   _get_fpu_type: proc

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

FPU_FLAG        equ 0001h
VME_FLAG        equ 0002h
DE_FLAG         equ 0004h
PSE_FLAG        equ 0008h
TSC_FLAG        equ 0010h
MSR_FLAG        equ 0020h
```

```
PAE_FLAG        equ 0040h
MCE_FLAG        equ 0080h
CX8_FLAG        equ 0100h
APIC_FLAG       equ 0200h
SEP_FLAG        equ 0800h
MTRR_FLAG       equ 1000h
PGE_FLAG        equ 2000h
MCA_FLAG        equ 4000h
CMOV_FLAG       equ 8000h
PAT_FLAG        equ 10000h
PSE36_FLAG      equ 20000h
PSNUM_FLAG      equ 40000h
CLFLUSH_FLAG    equ 80000h
DTS_FLAG        equ 200000h
ACPI_FLAG       equ 400000h
MMX_FLAG        equ 800000h
FXSR_FLAG       equ 1000000h
SSE_FLAG        equ 2000000h
SSE2_FLAG       equ 4000000h
SS_FLAG         equ 8000000h
HTT_FLAG        equ 10000000h
TM_FLAG         equ 20000000h
PBE_FLAG        equ 80000000h
SSE3_FLAG       equ 0001h
MONITOR_FLAG    equ 0008h
DS_CPL_FLAG     equ 0010h
EIST_FLAG       equ 0080h
TM2_FLAG        equ 0100h
CID_FLAG        equ 0400h
XTPR_FLAG       equ 04000h

EM64T_FLAG      equ 20000000h
XD_FLAG         equ 00100000h
SYSCALL_FLAG    equ 00000800h

.data
id_msg          db      "This system has a$"
cp_error        db      "n unknown processor$"
cp_8086         db      "n 8086/8088 processor$"
cp_286          db      "n 80286 processor$"
cp_386          db      "n 80386 processor$"

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

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

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

```
pentiumiimodel3_msg     db      " Genuine Intel(R) Pentium(R) II processor, model 3$"
pentiumiixeon_m5_msg    db      " Genuine Intel(R) Pentium(R) II processor, model 5 or"
                        db      " Intel(R) Pentium(R) II Xeon(TM) processor$"
pentiumiixeon_msg       db      " Genuine Intel(R) Pentium(R) II Xeon(TM) processor$"
celeron_msg             db      " Genuine Intel(R) Celeron(R) processor, model 5$"
celeronmodel6_msg       db      " Genuine Intel(R) Celeron(R) processor, model 6$"
celeron_brand           db      " Genuine Intel(R) Celeron(R) processor$"
pentiumiii_msg          db      " Genuine Intel(R) Pentium(R) III processor, model 7 or"
                        db      " Intel Pentium(R) III Xeon(TM) processor, model 7$"
pentiumiiixeon_msg      db      " Genuine Intel(R) Pentium(R) III Xeon(TM) processor, model 7$"
pentiumiiixeon_brand    db      " Genuine Intel(R) Pentium(R) III Xeon(TM) processor$"
pentiumiii_brand        db      " Genuine Intel(R) Pentium(R) III processor$"
mobile_piii_brand       db      " Genuine Mobile Intel(R) Pentium(R) III Processor-M$"
mobile_icp_brand        db      " Genuine Mobile Intel(R) Celeron(R) processor$"
mobile_P4_brand         db      " Genuine Mobile Intel(R) Pentium(R) 4 processor - M$"
pentium4_brand          db      " Genuine Intel(R) Pentium(R) 4 processor$"
xeon_brand              db      " Genuine Intel(R) Xeon(TM) processor$"
xeon_mp_brand           db      " Genuine Intel(R) Xeon(TM) processor MP$"
mobile_icp_brand_2      db      " Genuine Mobile Intel(R) Celeron(R) processor$"
mobile_pentium_m_brand  db      " Genuine Intel(R) Pentium(R) M processor$"
mobile_genuine_brand    db      " Mobile Genuine Intel(R) processor$"
mobile_icp_m_brand      db      " Genuine Intel(R) Celeron(R) M processor$"
unknown_msg             db      "n unknown Genuine Intel(R) processor$"

brand_entry     struct
        brand_value     db      ?
        brand_string    dw      ?
brand_entry     ends

brand_table     brand_entry     <01h, offset celeron_brand>
                brand_entry     <02h, offset pentiumiii_brand>
                brand_entry     <03h, offset pentiumiiixeon_brand>
                brand_entry     <04h, offset pentiumiii_brand>
                brand_entry     <06h, offset mobile_piii_brand>
                brand_entry     <07h, offset mobile_icp_brand>
                brand_entry     <08h, offset pentium4_brand>
                brand_entry     <09h, offset pentium4_brand>
                brand_entry     <0Ah, offset celeron_brand>
                brand_entry     <0Bh, offset xeon_brand>
                brand_entry     <0Ch, offset xeon_mp_brand>
                brand_entry     <0Eh, offset mobile_p4_brand>
                brand_entry     <0Fh, offset mobile_icp_brand>
                brand_entry     <11h, offset mobile_genuine_brand>
                brand_entry     <12h, offset mobile_icp_m_brand>
                brand_entry     <13h, offset mobile_icp_brand_2>
                brand_entry     <14h, offset celeron_brand>
                brand_entry     <15h, offset mobile_genuine_brand>
                brand_entry     <16h, offset mobile_pentium_m_brand>
                brand_entry     <17h, offset mobile_icp_brand_2>

brand_table_size    equ     ($ - offset brand_table) / (sizeof brand_entry)

; 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:        $"
ext_fam_msg     db      13,10," Extended Family: $"
ext_mod_msg     db      13,10," Extended Model:  $"
cr_lf           db      13,10,"$"
turbo_msg       db      13,10,"The processor is an OverDrive(R)"
                db      "  processor$"
dp_msg          db      13,10,"The processor is the upgrade"
                db      " processor in a dual processor system$"
fpu_msg         db      13,10,"The processor contains an on-chip"
                db      " FPU$"
vme_msg         db      13,10,"The processor supports Virtual"
                db      " Mode Extensions$"
de_msg          db      13,10,"The processor supports Debugging"
                db      " Extensions$"
pse_msg         db      13,10,"The processor supports Page Size"
                db      " Extensions$"
tsc_msg         db      13,10,"The processor supports Time Stamp"
                db      " Counter$"
msr_msg         db      13,10,"The processor supports Model"
                db      " Specific Registers$"
pae_msg         db      13,10,"The processor supports Physical"
                db      " Address Extensions$"
mce_msg         db      13,10,"The processor supports Machine"
                db      " Check Exceptions$"
cx8_msg         db      13,10,"The processor supports the"
                db      " CMPXCHG8B instruction$"
apic_msg        db      13,10,"The processor contains an on-chip"
                db      " APIC$"
sep_msg         db      13,10,"The processor supports Fast System"
                db      " Call$"
no_sep_msg      db      13,10,"The processor does not support Fast"
                db      " System Call$"
mtrr_msg        db      13,10,"The processor supports Memory Type"
                db      " Range Registers$"
pge_msg         db      13,10,"The processor supports Page Global"
                db      " Enable$"
mca_msg         db      13,10,"The processor supports Machine"
                db      " Check Architecture$"
cmov_msg        db      13,10,"The processor supports Conditional"
                db      " Move Instruction$"
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$"
```

```
clflush_msg     db      13,10,"The processor supports the"
                db      " CLFLUSH instruction$"
dts_msg         db      13,10,"The processor supports the"
                db      " Debug Trace Store feature$"
acpi_msg        db      13,10,"The processor supports the"
                db      " ACPI registers in MSR space$"
mmx_msg         db      13,10,"The processor supports Intel Architecture"
                db      " MMX(TM) Technology$"
fxsr_msg        db      13,10,"The processor supports Fast floating point"
                db      " save and restore$"
sse_msg         db      13,10,"The processor supports the Streaming"
                db      " SIMD extensions$"
sse2_msg        db      13,10,"The processor supports the Streaming"
                db      " SIMD extensions 2 instructions$"
ss_msg          db      13,10,"The processor supports Self-Snoop$"
htt_msg         db      13,10,"The processor supports Hyper-Threading Technology$"
tm_msg          db      13,10,"The processor supports the"
                db      " Thermal Monitor$"
pbe_msg         db      13,10,"The processor supports the"
                db      " Pending Break Event$"
sse3_msg        db      13,10,"The processor supports the Streaming SIMD"
                db      " Extensions 3 instructions$"
monitor_msg     db      13,10,"The processor supports the MONITOR and MWAIT"
                db      " instructions$"
ds_cpl_msg      db      13,10,"The processor supports Debug Store extensions for"
                db      " branch message storage by CPL$"
eist_msg        db      13,10,"The processor supports"
                db      " Enhanced SpeedStep(TM) Technology$"
tm2_msg         db      13,10,"The processor supports the"
                db      " Thermal Monitor 2$"
cid_msg         db      13,10,"The processor supports L1 Data Cache Context ID$"
xtpr_msg        db      13,10,"The processor supports transmitting TPR messages$"
em64t_msg       db      13,10,"The processor supports Intel(R) Extended Memory 64 Technology$"
xd_bit_msg      db      13,10,"The processor supports the Execute Disable Bit$"
syscall_msg     db      13,10,"The processor supports the SYSCALL & SYSRET instructions$"

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

.code
.8086

print   proc
```

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

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

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

print_86:
        cmp     _cpu_type, 0
        jne     print_286
        mov     dx, offset cp_8086
        mov     ah, 9h
        int     21h
        cmp     _fpu_type, 0
        je      end_print
        mov     dx, offset fp_8087
        mov     ah, 9h
        int     21h
        jmp     end_print

print_286:
        cmp     _cpu_type, 2
        jne     print_386
        mov     dx, offset cp_286
        mov     ah, 9h
        int     21h
        cmp     _fpu_type, 0
        je      end_print

print_287:
        mov     dx, offset fp_287
        mov     ah, 9h
        int     21h
        jmp     end_print

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

print_486:
        cmp     _cpu_type, 4
```

```
        jne     print_unknown           ; Intel processors will have
        mov     dx, offset cp_486sx     ;  CPUID instruction
        cmp     _fpu_type, 0
        je      print_486sx
        mov     dx, offset cp_486

print_486sx:
        mov     ah, 9h
        int     21h
        jmp     end_print

print_unknown:
        mov     dx, offset cp_error
        jmp     print_486sx

print_cpuid_data:
.486
        cmp     _intel_CPU, 1           ; check for genuine Intel
        jne     not_GenuineIntel        ;   processor

        mov     di, offset _brand_string ; brand string supported?
        cmp     byte ptr [di], 0
        je      print_brand_id

        mov     cx, 47                  ; max brand string length

skip_spaces:
        cmp     byte ptr [di], ' '      ; skip leading space chars
        jne     print_brand_string

        inc     di
        loop     skip_spaces

print_brand_string:
        cmp     cx, 0                   ; Nothing to print
        je      print_brand_id
        cmp     byte ptr [di], 0
        je      print_brand_id

print_brand_char:
        mov     dl, [di]                ; print upto the max chars
        mov     ah, 2
        int     21h

        inc     di
        cmp     byte ptr [di], 0
        je      print_family
        loop    print_brand_char
        jmp     print_family

print_brand_id:
        cmp     _cpu_type, 6
        jb      print_486_type
        ja      print_pentiumiiimodel8_type

        mov     eax, dword ptr _cpu_signature
        shr     eax, 4
        and     al, 0fh
        cmp     al, 8
        jae     print_pentiumiiimodel8_type
```

```
print_486_type:
        cmp     _cpu_type, 4                    ; if 4, print 80486 processor
        jne     print_pentium_type
        mov     eax, dword ptr _cpu_signature
        shr     eax, 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_unknown_type
        mov     eax, dword ptr _cpu_signature
        shr     eax, 4
        and     eax, 0fh                        ; isolate model
        cmp     eax, 3
        jge     print_pentiumiimodel3_type
        cmp     eax, 1
        jne     print_unknown_type              ; incorrect model number = 2
        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     dx, offset pentiumiimodel3_msg
        jmp     print_common

print_pentiumiimodel5_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:

; Is it Pentium II processor, model 5, Pentium II Xeon processor, Intel Celeron processor,
; Pentium III processor or Pentium III Xeon processor.

        mov     eax, dword ptr _cache_eax
        rol     eax, 8
        mov     cx, 3

celeron_detect_eax:
        cmp     al, 40h                         ; Is it no L2
        je      print_celeron_type
        cmp     al, 44h                         ; Is L2 >= 1M
```

```
        jae     print_pentiumiixeon_type

        rol     eax, 8
        loop    celeron_detect_eax

        mov     eax, dword ptr _cache_ebx
        mov     cx, 4

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

        rol     eax, 8
        loop    celeron_detect_ebx

        mov     eax, dword 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

        rol     eax, 8
        loop    celeron_detect_ecx

        mov     eax, dword 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

        rol     eax, 8
        loop    celeron_detect_edx

        mov     dx, offset pentiumiixeon_m5_msg
        mov     eax, dword ptr _cpu_signature
        shr     eax, 4
        and     eax, 0fh                ; isolate model
        cmp     eax, 5
        je      print_common
        mov     dx, offset pentiumiii_msg
        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
```

```
        je      print_common
        mov     dx, offset pentiumiiixeon_msg
        jmp     print_common

print_celeronmodel6_type:
        cmp     eax, 6                  ; if 6 & model 6, print Intel Celeron
                                        ; processor, model 6
        jne     print_pentiumiiimodel8_type
        mov     dx, offset celeronmodel6_msg
        jmp     print_common

print_pentiumiiimodel8_type:
        cmp     eax, 8                  ; Pentium III processor, model 8, or
                                        ; Pentium III Xeon processor, model 8
        jb      print_unknown_type

        mov     eax, dword ptr _features_ebx
        cmp     al, 0                   ; Is brand_id supported?
        je      print_unknown_type

        mov     di, offset brand_table  ; Setup pointer to brand_id table
        mov     cx, brand_table_size    ; Get maximum entry count

next_brand:
        cmp     al, byte ptr [di]       ; Is this the brand reported by the processor
        je      brand_found

        add     di, sizeof brand_entry  ; Point to next Brand Defined
        loop    next_brand              ; Check next brand if the table is not exhausted
        jmp     print_unknown_type

brand_found:
        mov     eax, dword ptr _cpu_signature
        cmp     eax, 06B1h              ; Check for Pentium III, model B, stepping 1
        jne     not_b1_celeron

        mov     dx, offset celeron_brand ; Assume this is a the special case (see Table 9)
        cmp     byte ptr[di], 3         ; Is this a B1 Celeron?
        je      print_common

not_b1_celeron:
        cmp     eax, 0F13h
        jae     not_xeon_mp

        mov     dx, offset xeon_mp_brand ; Early "Intel(R) Xeon(TM) processor MP"?
        cmp     byte ptr [di], 0Bh
        je      print_common

        mov     dx, offset xeon_brand   ; Early "Intel(R) Xeon(TM) processor"?
        cmp     byte ptr[di], 0Eh
        je      print_common

not_xeon_mp:
        mov     dx, word ptr [di+1]     ; Load DX with the offset of the brand string
        jmp     print_common

print_unknown_type:
        mov     dx, offset unknown_msg  ; if neither, print unknown
print_common:
        mov     ah, 9h
```

```
        int     21h

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

        mov     eax, dword ptr _cpu_signature
        and     ah, 0fh                 ; Check for Extended Family
        cmp     ah, 0fh
        jne     print_model
        mov     dx, offset ext_fam_msg
        mov     ah, 9h
        int     21h
        shr     eax, 20
        mov     ah, al                  ; Copy extended family into ah
        shr     al, 4
        and     ax, 0f0fh
        add     ah, '0'                 ; Convert upper nibble to ascii
        add     al, '0'                 ; Convert lower nibble to ascii
        push    ax
        mov     dl, al
        mov     ah, 2
        int     21h                     ; print upper nibble of ext family
        pop     ax
        mov     dl, ah
        mov     ah, 2
        int     21h                     ; print lower nibble of ext family

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

         mov    eax, dword ptr _cpu_signature
        and     al, 0f0h                ; Check for Extended Model
        cmp     ah, 0f0h
        jne     print_stepping
        mov     dx, offset ext_mod_msg
        mov     ah, 9h
        int     21h
        shr     eax, 16
        and     al, 0fh
        add     al, '0'                 ; Convert extended model to ascii
        mov     dl, al
        mov     ah, 2
        int     21h                     ; print lower nibble of ext family

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

print_upgrade:
        mov     eax, dword 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     eax, dword ptr _features_edx
        and     eax, FPU_FLAG           ; check for FPU
        jz      check_VME
        mov     dx, offset fpu_msg
        mov     ah, 9h
        int     21h

check_VME:
        mov     eax, dword ptr _features_edx
        and     eax, VME_FLAG           ; check for VME
        jz      check_DE
        mov     dx, offset vme_msg
        mov     ah, 9h
        int     21h

check_DE:
        mov     eax, dword ptr _features_edx
        and     eax, DE_FLAG            ; check for DE
        jz      check_PSE
        mov     dx, offset de_msg
        mov     ah, 9h
        int     21h

check_PSE:
        mov     eax, dword ptr _features_edx
        and     eax, PSE_FLAG           ; check for PSE
        jz      check_TSC
        mov     dx, offset pse_msg
        mov     ah, 9h
        int     21h

check_TSC:
        mov     eax, dword ptr _features_edx
        and     eax, TSC_FLAG           ; check for TSC
        jz      check_MSR
        mov     dx, offset tsc_msg
        mov     ah, 9h
        int     21h

check_MSR:
        mov     eax, dword ptr _features_edx
        and     eax, MSR_FLAG           ; check for MSR
        jz      check_PAE
        mov     dx, offset msr_msg
        mov     ah, 9h
        int     21h

check_PAE:
        mov     eax, dword ptr _features_edx
```

```
        and     eax, PAE_FLAG           ; check for PAE
        jz      check_MCE
        mov     dx, offset pae_msg
        mov     ah, 9h
        int     21h

check_MCE:
        mov     eax, dword ptr _features_edx
        and     eax, MCE_FLAG           ; check for MCE
        jz      check_CX8
        mov     dx, offset mce_msg
        mov     ah, 9h
        int     21h

check_CX8:
        mov     eax, dword ptr _features_edx
        and     eax, CX8_FLAG           ; check for CMPXCHG8B
        jz      check_APIC
        mov     dx, offset cx8_msg
        mov     ah, 9h
        int     21h

check_APIC:
        mov     eax, dword ptr _features_edx
        and     eax, APIC_FLAG          ; check for APIC
        jz      check_SEP
        mov     dx, offset apic_msg
        mov     ah, 9h
        int     21h

check_SEP:
        mov     eax, dword ptr _features_edx
        and     eax, SEP_FLAG           ; Check for Fast System Call
        jz      check_MTRR

        cmp     _cpu_type, 6            ; Determine if Fast System
        jne     print_sep               ;  Calls are supported.

        mov     eax, dword ptr _cpu_signature
        cmp     al, 33h
        jb      print_no_sep

print_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     eax, dword ptr _features_edx
        and     eax, MTRR_FLAG          ; check for MTRR
        jz      check_PGE
        mov     dx, offset mtrr_msg
        mov     ah, 9h
        int     21h
```

```
check_PGE:
        mov     eax, dword ptr _features_edx
        and     eax, PGE_FLAG           ; check for PGE
        jz      check_MCA
        mov     dx, offset pge_msg
        mov     ah, 9h
        int     21h

check_MCA:
        mov     eax, dword ptr _features_edx
        and     eax, MCA_FLAG           ; check for MCA
        jz      check_CMOV
        mov     dx, offset mca_msg
        mov     ah, 9h
        int     21h

check_CMOV:
        mov     eax, dword ptr _features_edx
        and     eax, 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
        jz      check_PSE36
        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_CLFLUSH
        mov     dx, offset psnum_msg
        mov     ah, 9h
        int     21h

check_CLFLUSH:
        mov     eax, dword ptr _features_edx
        and     eax, CLFLUSH_FLAG       ; check for Cache Line Flush
        jz      check_DTS
        mov     dx, offset clflush_msg
        mov     ah, 9h
        int     21h

check_DTS:
        mov     eax, dword ptr _features_edx
        and     eax, DTS_FLAG           ; check for Debug Trace Store
        jz      check_ACPI
```

```
        mov     dx, offset dts_msg
        mov     ah, 9h
        int     21h

check_ACPI:
        mov     eax, dword ptr _features_edx
        and     eax, ACPI_FLAG          ; check for processor serial number
        jz      check_MMX
        mov     dx, offset acpi_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_SSE
        mov     dx, offset fxsr_msg
        mov     ah, 9h
        int     21h

check_SSE:
        mov     eax, dword ptr _features_edx
        and     eax, SSE_FLAG           ; check for Streaming SIMD
        jz      check_SSE2              ;   Extensions
        mov     dx, offset sse_msg
        mov     ah, 9h
        int     21h

check_SSE2:
        mov     eax, dword ptr _features_edx
        and     eax, SSE2_FLAG          ; check for Streaming SIMD
        jz      check_SS                ;   Extensions 2
        mov     dx, offset sse2_msg
        mov     ah, 9h
        int     21h

check_SS:
        mov     eax, dword ptr _features_edx
        and     eax, SS_FLAG            ; check for Self Snoop
        jz      check_HTT
        mov     dx, offset ss_msg
        mov     ah, 9h
        int     21h

check_HTT:
        mov     eax, dword ptr _features_edx
        and     eax, HTT_FLAG           ; check for Hyper-Thread Technology
        jz      check_TM

        mov     eax, dword ptr _features_ebx
        bswap   eax                     ; Put Logical processor count in reg AH
        cmp     ah, 1                   ; Logical processor count > 1?
```

```
        je      check_TM

        mov     dx, offset htt_msg          ; Supports HTT
        mov     ah, 9h
        int     21h

check_TM:
        mov     eax, dword ptr _features_edx
        and     eax, TM_FLAG                ; check for Thermal Monitor
        jz      check_PBE
        mov     dx, offset tm_msg
        mov     ah, 9h
        int     21h

check_PBE:
        mov     eax, dword ptr _features_edx
        and     eax, PBE_FLAG               ; check for Pending Break Event
        jz      check_sse3
        mov     dx, offset pbe_msg
        mov     ah, 9h
        int     21h

check_sse3:
        mov     eax, dword ptr _features_ecx
        and     eax, SSE3_FLAG              ; check for SSE3 instructions
        jz      check_monitor
        mov     dx, offset sse3_msg
        mov     ah, 9h
        int     21h

check_monitor:
        mov     eax, dword ptr _features_ecx
        and     eax, MONITOR_FLAG           ; check for monitor/mwait instructions
        jz      check_ds_cpl
        mov     dx, offset monitor_msg
        mov     ah, 9h
        int     21h

check_ds_cpl:
        mov     eax, dword ptr _features_ecx
        and     eax, DS_CPL_FLAG            ; check for Debug Store extensions qualified by CPL
        jz      check_EIST
        mov     dx, offset ds_cpl_msg
        mov     ah, 9h
        int     21h

check_EIST:
        mov     eax, dword ptr _features_ecx
        and     eax, EIST_FLAG              ; check for Enhanced SpeedStep Technology
        jz      check_TM2
        mov     dx, offset eist_msg
        mov     ah, 9h
        int     21h

check_TM2:
        mov     eax, dword ptr _features_ecx
        and     eax, TM2_FLAG               ; check for Thermal Monitor 2
        jz      check_CID
        mov     dx, offset tm2_msg
        mov     ah, 9h
```

```
        int     21h

check_CID:
        mov     eax, dword ptr _features_ecx
        and     eax, CID_FLAG           ; check for L1 Context ID
        jz      check_XTPR
        mov     dx, offset cid_msg
        mov     ah, 9h
        int     21h

check_XTPR:
        mov     eax, dword ptr _features_ecx
        and     eax, XTPR_FLAG          ; check for echo Task Priority
        jz      check_SYSCALL
        mov     dx, offset xtpr_msg
        mov     ah, 9h
        int     21h

check_SYSCALL:
        mov     eax, dword ptr _ext_funct_1_edx
        and     eax, SYSCALL_FLAG       ; check for SYSCALL/SYSRET instructions
        jz      check_XD
        mov     dx, offset syscall_msg
        mov     ah, 9h
        int     21h

check_XD:
        mov     eax, dword ptr _ext_funct_1_edx
        and     eax, XD_FLAG            ; check for echo Task Priority
        jz      check_EM64T
        mov     dx, offset xd_bit_msg
        mov     ah, 9h
        int     21h

check_EM64T:
        mov     eax, dword ptr _ext_funct_1_edx
        and     eax, EM64T_FLAG         ; check for echo Task Priority
        jz      end_print
        mov     dx, offset em64t_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
```

例 3. C 言語で記述されたプロセッサ識別プロシージャ

```
/* FILENAME:  CPUID3.C                                                    */
/* Copyright (c) Intel Corporation 1994-2004                              */
/*                                                                        */
/* 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(R) II, Pentium(R) II Xeon(TM), Pentium(R) II OverDrive(R),     */
/* Intel(R) Celeron(R), Pentium(R) III processors, Pentium(R) III Xeon(TM) */
/* processors, Pentium(R) 4 processors and Intel(R) Xeon(TM) processors   */

#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         0x1000
#define PGE_FLAG          0x2000
#define MCA_FLAG          0x4000
#define CMOV_FLAG         0x8000
#define PAT_FLAG          0x10000
#define PSE36_FLAG        0x20000
#define PSNUM_FLAG        0x40000
#define CLFLUSH_FLAG      0x80000
#define DTS_FLAG          0x200000
#define ACPI_FLAG         0x400000
#define MMX_FLAG          0x800000
#define FXSR_FLAG         0x1000000
#define SSE_FLAG          0x2000000
#define SSE2_FLAG         0x4000000
```

```
#define SS_FLAG         0x8000000
#define HTT_FLAG        0x10000000
#define TM_FLAG         0x20000000
#define PBE_FLAG        0x80000000
#define SSE3_FLAG       0x0001
#define MONITOR_FLAG    0x0008
#define DS_CPL_FLAG     0x0010
#define EIST_FLAG       0x0080
#define TM2_FLAG        0x0100
#define CID_FLAG        0x0400
#define XTPR_FLAG       0x4000

#define SYSCALL_FLAG    0x00000800
#define XD_FLAG         0x00100000
#define EM64T_FLAG      0x20000000


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;
extern char brand_string[48];
extern int  brand_id;

long cache_temp;
long celeron_flag;
long pentiumxeon_flag;

struct brand_entry {
   long    brand_value;
   char    *brand_string;
};

#define brand_table_size 15

struct brand_entry brand_table[brand_table_size] = {
   0x01, " Genuine Intel(R) Celeron(R) processor",
   0x02, " Genuine Intel(R) Pentium(R) III processor",
   0x03, " Genuine Intel(R) Pentium(R) III Xeon(TM) processor",
   0x04, " Genuine Intel(R) Pentium(R) III processor",
   0x06, " Genuine Mobile Intel(R) Pentium(R) III Processor - M",
   0x07, " Genuine Mobile Intel(R) Celeron(R) processor",
   0x08, " Genuine Intel(R) Pentium(R) 4 processor",
   0x09, " Genuine Intel(R) Pentium(R) 4 processor",
   0x0A, " Genuine Intel(R) Celeron(R) processor",
   0x0B, " Genuine Intel(R) Xeon(TM) processor",
   0x0C, " Genuine Intel(R) Xeon(TM) Processor MP",
   0x0E, " Genuine Mobile Intel(R) Pentium(R) 4 Processor - M",
   0x0F, " Genuine Mobile Intel(R) Celeron(R) processor",
   0x11, " Mobile Genuine Intel(R) processor",
   0x12, " Genuine Mobile Intel(R) Celeron(R) M processor",
```

```
    0x13, " Genuine Mobile Intel(R) Celeron(R) processor",
    0x14, " Genuine Intel(R) Celeron(R) processor",
    0x15, " Mobile Genuine Intel(R) processor",
    0x16, " Genuine Intel(R) Pentium(R) M processor"
    0x17, " Genuine Mobile Intel(R) Celeron(R) processor",
};


int main() {
    get_cpu_type();
    get_fpu_type();
    print();
    return(0);
}

int print() {
    int brand_index = 0;

    printf("This system has a");
    if (cpuid_flag == 0) {
        switch (cpu_type) {
        case 0:
            printf("n 8086/8088 processor");
            if (fpu_type) printf(" and an 8087 math coprocessor");
            break;
        case 2:
            printf("n 80286 processor");
            if (fpu_type) printf(" and an 80287 math coprocessor");
            break;
        case 3:
            printf("n 80386 processor");
            if (fpu_type == 2)
                printf(" and an 80287 math coprocessor");
            else if (fpu_type)
                printf(" and an 80387 math coprocessor");
            break;
        case 4:
            if (fpu_type)
                printf("n 80486DX, 80486DX2 processor or 80487SX math coprocessor");
            else
                printf("n 80486SX processor");
            break;
        default:
            printf("n unknown processor");
        }
    }
    else {
    /* using cpuid instruction */
        if (intel_CPU) {
            if (brand_string[0]) {
                brand_index = 0;
                while ((brand_string[brand_index] == ' ') && (brand_index < 48))
                    brand_index++;
                if (brand_index != 48)
                    printf(" %s", &brand_string[brand_index]);
            }
            else if (cpu_type == 4) {
                switch ((cpu_signature>>4) & 0xf) {
                case  0:
                case  1:
```

```
      printf(" Genuine Intel486(TM) DX processor");
      break;
   case  2:
      printf(" Genuine Intel486(TM) SX processor");
      break;
   case  3:
      printf(" Genuine IntelDX2(TM) processor");
      break;
   case  4:
      printf(" Genuine Intel486(TM) processor");
      break;
   case  5:
      printf(" Genuine IntelSX2(TM) processor");
      break;
   case  7:
      printf(" Genuine Write-Back Enhanced \
         IntelDX2(TM) processor");
      break;
   case  8:
      printf(" Genuine IntelDX4(TM) processor");
      break;
   default:
      printf(" Genuine Intel486(TM) processor");
   }
}
else if (cpu_type == 5)
   printf(" Genuine Intel Pentium(R) processor");
else if ((cpu_type == 6) && (((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)) ||
         ((cpu_type == 6) && (((cpu_signature >> 4) & 0xf) == 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))
      pentiumxeon_flag = 1;

   cache_temp = cache_eax & 0xFF0000;
   if (cache_temp == 0x400000)
      celeron_flag = 1;
   if ((cache_temp >= 0x440000) && (cache_temp <= 0x450000))
      pentiumxeon_flag = 1;

   cache_temp = cache_eax & 0xFF00;
   if (cache_temp == 0x4000)
      celeron_flag = 1;
   if ((cache_temp >= 0x4400) && (cache_temp <= 0x4500))
      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 & 0xFF0000;
if (cache_temp == 0x400000)
   celeron_flag = 1;
if ((cache_temp >= 0x440000) && (cache_temp <= 0x450000))
   pentiumxeon_flag = 1;

cache_temp = cache_ebx & 0xFF00;
if (cache_temp == 0x4000)
   celeron_flag = 1;
if ((cache_temp >= 0x4400) && (cache_temp <= 0x4500))
   pentiumxeon_flag = 1;

cache_temp = cache_ebx & 0xFF;
if (cache_temp == 0x40)
   celeron_flag = 1;
if ((cache_temp >= 0x44) && (cache_temp <= 0x45))
   pentiumxeon_flag = 1;

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

cache_temp = cache_ecx & 0xFF0000;
if (cache_temp == 0x400000)
   celeron_flag = 1;
if ((cache_temp >= 0x440000) && (cache_temp <= 0x450000))
   pentiumxeon_flag = 1;

cache_temp = cache_ecx & 0xFF00;
if (cache_temp == 0x4000)
   celeron_flag = 1;
if ((cache_temp >= 0x4400) && (cache_temp <= 0x4500))
   pentiumxeon_flag = 1;

cache_temp = cache_ecx & 0xFF;
if (cache_temp == 0x40)
   celeron_flag = 1;
if ((cache_temp >= 0x44) && (cache_temp <= 0x45))
   pentiumxeon_flag = 1;

cache_temp = cache_edx & 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 >= 0x440000) && (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(R) 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,");
            printf(" model 7");
      }
      else {
         if (((cpu_signature >> 4) & 0x0f) == 5) {
            printf(" Genuine Intel Pentium(R) II processor, model 5 ");
            printf("or Intel Pentium(R) II Xeon(TM) processor");
         }
         else {
            printf(" Genuine Intel Pentium(R) III processor, model 7");
            printf(" or Intel Pentium(R) III Xeon(TM) processor,");
            printf(" model 7");
         }
      }
   }
}
else if ((cpu_type == 6) && (((cpu_signature >> 4) & 0xf) == 6))
   printf(" Genuine Intel Celeron(R) processor, model 6");
else if ((features_ebx & 0xff) != 0) {
   while ((brand_index < brand_table_size) &&
      ((features_ebx & 0xff) != brand_table[brand_index].brand_value))
      brand_index++;
   if (brand_index < brand_table_size) {
      if ((cpu_signature == 0x6B1) &&
         (brand_table[brand_index].brand_value == 0x3))
         printf(" Genuine Intel(R) Celeron(R) processor");
      else if ((cpu_signature < 0xF13) &&
         (brand_table[brand_index].brand_value == 0x0B))
         printf(" Genuine Intel(R) Xeon(TM) processor MP");
      else if ((cpu_signature < 0xF13) &&
         (brand_table[brand_index].brand_value == 0x0E))
         printf(" Genuine Intel(R) Xeon(TM) processor");
      else
         printf("%s", brand_table[brand_index].brand_string);
   }
   else
      printf("n unknown Genuine Intel processor");
}
else
   printf("n unknown Genuine Intel processor");
printf("\nProcessor Family: %X", cpu_type);
if (cpu_type == 0xf)
   printf("\n Extended Family: %x",(cpu_signature>>20)&0xff);
printf("\nModel:           %X", (cpu_signature>>4)&0xf);
if (((cpu_signature>>4) & 0xf) == 0xf)
   printf("\n Extended Model:  %x",(cpu_signature>>16)&0xf);
```

```
printf("\nStepping:        %X\n", 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("\nThe processor supports the Debugging Extensions");
if (features_edx & PSE_FLAG)
   printf("\nThe processor supports Page Size Extensions");
if (features_edx & TSC_FLAG)
   printf("\nThe processor supports Time Stamp Counter");
if (features_edx & MSR_FLAG)
   printf("\nThe processor supports Model Specific Registers");
if (features_edx & PAE_FLAG)
   printf("\nThe processor supports Physical Address Extension");
if (features_edx & MCE_FLAG)
   printf("\nThe processor supports Machine Check Exceptions");
if (features_edx & CX8_FLAG)
   printf("\nThe processor supports the CMPXCHG8B instruction");
if (features_edx & APIC_FLAG)
   printf("\nThe processor contains an on-chip APIC");
if (features_edx & SEP_FLAG) {
   if ((cpu_type == 6) && ((cpu_signature & 0xff) < 0x33))
      printf("\nThe processor does not support the Fast System Call");
   else
      printf("\nThe processor supports the Fast System Call");
}
if (features_edx & MTRR_FLAG)
   printf("\nThe processor supports the Memory Type Range Registers");
if (features_edx & PGE_FLAG)
   printf("\nThe processor supports Page Global Enable");
if (features_edx & MCA_FLAG)
   printf("\nThe processor supports the Machine Check Architecture");
if (features_edx & CMOV_FLAG)
   printf("\nThe processor supports the Conditional Move Instruction");
if (features_edx & PAT_FLAG)
   printf("\nThe processor supports the Page Attribute Table");
if (features_edx & PSE36_FLAG)
   printf("\nThe processor supports 36-bit Page Size Extension");
if (features_edx & PSNUM_FLAG)
   printf("\nThe processor supports the processor serial number");
if (features_edx & CLFLUSH_FLAG)
   printf("\nThe processor supports the CLFLUSH instruction");
if (features_edx & DTS_FLAG)
   printf("\nThe processor supports the Debug Trace Store feature");
if (features_edx & ACPI_FLAG)
   printf("\nThe processor supports ACPI registers in MSR space");
if (features_edx & MMX_FLAG)
   printf("\nThe processor supports Intel Architecture MMX(TM) technology");
if (features_edx & FXSR_FLAG)
   printf("\nThe processor supports the Fast floating point save and restore");
if (features_edx & SSE_FLAG)
   printf("\nThe processor supports the Streaming SIMD extensions to the Intel Architecture");
if (features_edx & SSE2_FLAG)
   printf("\nThe processor supports the Streaming SIMD extensions 2 instructions");
if (features_edx & SS_FLAG)
```

```
            printf("\nThe processor supports Self-Snoop");
         if ((features_edx & HTT_FLAG) &&
            (((features_ebx >> 16) & 0x0FF) > 1))
            printf("\nThe processor supports Hyper-Threading Technology");
         if (features_edx & TM_FLAG)
            printf("\nThe processor supports the Thermal Monitor");
         if (features_edx & PBE_FLAG)
            printf("\nThe processor supports Pending Break Event signaling");
         if (features_ecx & SSE3_FLAG)
            printf("\nThe processor supports the Streaming SIMD extensions 3 instructions");
         if (features_ecx & MONITOR_FLAG)
            printf("\nThe processor supports the MONITOR and MWAIT instructions");
         if (features_ecx & DS_CPL_FLAG)
            printf("\nThe processor supports Debug Store extensions for branch message storage by CPL");
         if (features_ecx & EIST_FLAG)
            printf("\nThe processor supports Enhanced SpeedStep(TM) Technology");
         if (features_ecx & TM2_FLAG)
            printf("\nThe processor supports the Thermal Monitor 2");
         if (features_ecx & CID_FLAG)
            printf("\nThe processor supports L1 Data Cache Context ID");
         if (features_ecx & XTPR_FLAG)
            printf("\nThe processor supports transmitting TPR messages");
         if (ext_funct_1_edx & SYSCALL_FLAG)
            printf("\nThe processor supports the SYSCALL & SYSRET instructions");
         if (ext_funct_1_edx & XD_FLAG)
            printf("\nThe processor supports the Execute Disable Bit ");
         if (ext_funct_1_edx & EM64T_FLAG)
            printf("\nThe processor supports Intel(R) Extended Memory 64 Technology ");
      }
      else {
         printf("t least an 80486 processor. ");
         printf("\nIt does not contain a Genuine Intel part and as a result, the ");
         printf("\nCPUID detection information cannot be determined at this time.");
      }
   }
   printf("\n");
   return(0);
}
```

例 4. 例外ハンドラを使用した拡張命令の検出

```
// FILENAME:  FEATURES.CPP
// Copyright (c) Intel Corporation 2000-2004
//
// 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.
//
#include "stdio.h"
#include "string.h"
#include "excpt.h"

    // The follow code sample demonstrate using exception handlers to identify available IA-32 features,
    // The sample code Identifies IA-32 features such as support for Streaming SIMD Extensions 3,
    // Streaming SIMD Extensions 2 (SSE2), support for Streaming SIMD Extensions (SSE),
    // support for MMX (TM) instructions.
    // This technique can be used safely to determined IA-32 features and provide
    // forward compatibility to run optimally on future IA-32 processors.
    // Please note that the technique of trapping invalid opcodes is not suitable
    // for identifying the processor family and model.

int main(int argc, char* argv[])
{
    char  sSupportSSE3[80]="Don't know";
    char  sSupportSSE2[80]="Don't know";
    char  sSupportSSE[80]="Don't know";
    char  sSupportMMX[80]="Don't know";

    // To identify whether SSE3, SSE2, SSE, or MMX instructions are supported on an x86 compatible
    // processor in a fashion that will be compatible to future IA-32 processors,
    // The following tests are performed in sequence: (This sample code will assume cpuid
    //          instruction is supported by the target processor.)
    // 1. Test whether target processor is a Genuine Intel processor, if yes
    // 2. Test if executing an SSE3 instruction would cause an exception, if no exception occurs,
    //          SSE3 is supported; if exception occurs,
    // 3. Test if executing an SSE2 instruction would cause an exception, if no exception occurs,
    //          SSE2 is supported; if exception occurs,
    // 4. Test if executing an SSE instruction would cause an exception, if no exception occurs,
    //          SSE is supported; if exception occurs,
    // 5. Test if executing an MMX instruction would cause an exception, if no exception occurs,
    //          MMX instruction is supported,
    //          if exception occurs, MMX instruction is not supported by this processor.

    // For clarity, the following stub function "IsGenuineIntelProcessor()" is not shown in this example,

    // The function "IsGenuineIntelProcessor()" can be adapted from the sample code implementation
    // of the assembly procedure "_get_cpu_type". The purpose of this stub function is to examine
    // whether the Vendor ID string, which is returned when executing
    // cpuid instruction with EAX = 0, indicates the processor is a genuine Intel processor.
```

```
if (IsGenuineIntelProcessor())
{
    // First, execute an SSE3 instruction to see whether an exception occurs

    __try
    {
        __asm {
            haddpd xmm1, xmm2           // this is an instruction available in SSE3
            // _emit 0x66 __asm _emit 0x0F __asm _emit 0x7C __asm _emit 0xCA

        strcpy(&sSupportSSE3[0], "Yes");            // No exception executing an SSE3 instruction
    }

    __except( EXCEPTION_EXECUTE_HANDLER  ) // SSE3 exception handler
    {
        // exception occurred when executing an SSE3 instruction
        strcpy(&sSupportSSE3[0], "No");
    }

    // Second, execute an SSE2 instruction to see whether an exception occurs

    __try
    {
        __asm {
            paddq xmm1, xmm2            // this is an instruction available in SSE2
        }
        strcpy(&sSupportSSE2[0], "Yes");            // No exception executing an SSE2 instruction
    }

    __except( EXCEPTION_EXECUTE_HANDLER  ) // SSE2 exception handler
    {
        // exception occurred when executing an SSE2 instruction
        strcpy(&sSupportSSE2[0], "No");
    }

    // Third, execute an SSE instruction to see whether an exception occurs

    __try
    {
        __asm {
            orps xmm1, xmm2 // this is an instruction available in SSE
            //__asm _emit 0x66 __asm _emit 0x0f __asm _emit 0x57 __asm _emit 0xc0
        }
        strcpy(&sSupportSSE[0], "Yes"); // no exception executing an SSE instruction
    }

    __except( EXCEPTION_EXECUTE_HANDLER  )          // SSE exception handler
    {
        // exception occurred when executing an SSE instruction
        strcpy(&sSupportSSE[0], "No");
    }

    // Fourth, execute an MMX instruction to see whether an exception occurs

    __try
    {
        __asm {
            emms    // this is an instruction available in MMX          }
        strcpy(&sSupportMMX[0], "Yes");   // no exception executing an MMX instruction
```

```
            }

            __except( EXCEPTION_EXECUTE_HANDLER  )// MMX exception handler
            {
                // exception occurred when executing an MMX instruction
                strcpy(&sSupportMMX[0], "No");
            }
        }

        printf("This Processor supports the following instruction extensions: \n");
        printf("SSE3 instruction: \t\t%s \n", &sSupportSSE3[0]);
        printf("SSE2 instruction: \t\t%s \n", &sSupportSSE2[0]);
        printf("SSE instruction: \t\t%s \n", &sSupportSSE[0]);
        printf("MMX instruction: \t\t%s \n", &sSupportMMX[0]);
        return 0;
}
```

例 5. デノーマル・ゼロのサポートの検出

```
;       Filename:  DAZDTECT.ASM
;       Copyright (c) Intel Corporation 2001-2004
;
;       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 example assumes the system has booted DOS.
;       This program runs in Real mode.
;
;*****************************************************************
;
;       This program was assembled using MASM 6.14.8444.
;
;       This program performs the following 8 steps to determine if the
;       processor supports the SSE/SSE2 DAZ mode.
;
; Step 1.  Execute the CPUID instruction with an input value of EAX=0 and
;       ensure the vendor-ID string returned is "GenuineIntel".
;
; Step 2.  Execute the CPUID instruction with EAX=1. This will load the
;       EDX register with the feature flags.
;
; Step 3.  Ensure that the FXSR feature flag (EDX bit 24) is set.
;       This indicates the processor supports the FXSAVE and FXRSTOR
;       instructions.
;
; Step 4.  Ensure that the XMM feature flag (EDX bit 25) or the EMM feature
;       flag (EDX bit 26) is set. This indicates that the processor supports
;       at least one of the SSE/SSE2 instruction sets and its MXCSR control
;       register.
;
; Step 5.  Zero a 16-byte aligned, 512-byte area of memory.
;       This is necessary since some implementations of FXSAVE do not
;       modify reserved areas within the image.
;
; Step 6.  Execute an FXSAVE into the cleared area.
;
; Step 7.  Bytes 28-31 of the FXSAVE image are defined to contain the
;       MXCSR_MASK.  If this value is 0, then the processor's MXCSR_MASK
;       is 0xFFBF, otherwise MXCSR_MASK is the value of this dword.
;
; Step 8.  If bit 6 of the MXCSR_MASK is set, then DAZ is supported.
;
;*****************************************************************

        .DOSSEG
        .MODEL small, c
```

```
        .STACK


; Data segment

        .DATA

buffer          DB      512+16 DUP (0)

not_intel       DB      "This is not an Genuine Intel processor.", 0Dh, 0Ah, "$"
noSSEorSSE2     DB      "Neither SSE or SSE2 extensions are supported.", 0Dh, 0Ah, "$"
no_FXSAVE       DB      "FXSAVE not supported.", 0Dh, 0Ah, "$"
daz_mask_clear  DB      "DAZ bit in MXCSR_MASK is zero (clear).", 0Dh, 0Ah, "$"
no_daz          DB      "DAZ mode not supported.", 0Dh, 0Ah, "$"
supports_daz    DB      "DAZ mode supported.", 0Dh, 0Ah, "$"


; Code segment

        .CODE
        .686p
        .XMM

dazdtect PROC NEAR

        .startup                        ; Allow assembler to create code that
                                        ; initializes stack and data segment
                                        ; registers

; Step 1.

        ;Verify Genuine Intel processor by checking CPUID generated vendor ID

        mov     eax, 0
        cpuid

        cmp     ebx, 'uneG'             ; Compare first 4 letters of Vendor ID
        jne     notIntelprocessor       ; Jump if not Genuine Intel processor
        cmp     edx, 'Ieni'             ; Compare next 4 letters of Vendor ID
        jne     notIntelprocessor       ; Jump if not Genuine Intel processor
        cmp     ecx, 'letn'             ; Compare last 4 letters of Vendor ID
        jne     notIntelprocessor       ; Jump if not Genuine Intel processor

; Step 2, 3, and 4

        ; Get CPU feature flags
        ; Verify FXSAVE and either SSE or
        ; SSE2 are supported

        mov     eax, 1
        cpuid
        bt      edx, 24t                ; Feature Flags Bit 24 is FXSAVE support
        jnc     noFxsave                ; jump if FXSAVE not supported

        bt      edx, 25t                ; Feature Flags Bit 25 is SSE support
        jc      sse_or_sse2_supported   ; jump if SSE is not supported

        bt      edx, 26t                ; Feature Flags Bit 26 is SSE2 support
        jnc     no_sse_sse2             ; jump if SSE2 is not supported
```

```
sse_or_sse2_supported:

        ; FXSAVE requires a 16-byte aligned
        ; buffer so get offset into buffer

        mov     bx, OFFSET buffer       ; Get offset of the buffer into bx
        and     bx, 0FFF0h
        add     bx, 16t                 ; DI is aligned at 16-byte boundary

; Step 5.

        ; Clear the buffer that will be
        ; used for FXSAVE data

        push    ds
        pop     es
        mov     di, bx
        xor     ax, ax
        mov     cx, 512/2
        cld
        rep     stosw                   ; Fill at FXSAVE buffer with zeroes

; Step 6.

        fxsave  [bx]

; Step 7.

        mov     eax, DWORD PTR [bx][28t] ; Get MXCSR_MASK
        cmp     eax, 0                  ; Check for valid mask
        jne     check_mxcsr_mask
        mov     eax, 0FFBFh             ; Force use of default MXCSR_MASK

check_mxcsr_mask:
; EAX contains MXCSR_MASK from FXSAVE buffer or default mask

; Step 8.

        bt      eax, 6t                 ; MXCSR_MASK Bit 6 is DAZ support
        jc      supported               ; Jump if DAZ supported

        mov     dx, OFFSET daz_mask_clear
        jmp     notSupported

supported:
        mov     dx, OFFSET supports_daz ; Indicate DAZ is supported.
        jmp     print

notIntelProcessor:
        mov     dx, OFFSET not_intel    ; Assume not an Intel processor
        jmp     print

no_sse_sse2:
        mov     dx, OFFSET noSSEorSSE2  ; Setup error message assuming no SSE/SSE2
        jmp     notSupported

noFxsave:
        mov     dx, OFFSET no_FXSAVE

notSupported:
```

```
                mov     ah, 09h                 ; Execute DOS print string function
                int     21h

                mov     dx, OFFSET no_daz

print:
                mov     ah, 09h                 ; Execute DOS print string function
                int     21h

exit:
                .exit                           ; Allow assembler to generate code
                                                ; that returns control to DOS
                ret

dazdtect        ENDP

                END
```

例 6. 周波数の計算

```
;           Filename:  FREQUENC.ASM
;           Copyright (c) Intel Corporation 2001-2004
;
;           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 example assumes the system has booted DOS.
;           This program runs in Real mode.
;
;*****************************************************************************
;
;           This program was assembled using MASM 6.14.8444 and tested on a
;           system with a Pentium(r) II processor, a system with a
;           Pentium(r) III processor, a system with a Pentium(r) 4 processor,
;           B2 stepping, and a system with a Pentium(r) 4 processor,
;           C1 stepping.
;
;           This program performs the following 8 steps to determine the
;           actual processor frequency.
;
; Step 1.  Execute the CPUID instruction with an input value of EAX=0
;           and ensure the vendor-ID string returned is "GenuineIntel".
; Step 2.  Execute the CPUID instruction with EAX=1 to load the EDX
;           register with the feature flags.
; Step 3.  Ensure that the TSC feature flag (EDX bit 4) is set. This
;           indicates the processor supports the Time-Stamp Counter
;           and RDTSC instruction.
; Step 4.  Read the TSC at the beginning of the reference period
; Step 5.  Read the TSC at the end of the reference period.
; Step 6.  Compute the TSC delta from the beginning and ending of the
;           reference period.
; Step 7.  Compute the actual frequency by dividing the TSC delta by
;           the reference period.
;
;*****************************************************************************

            .DOSSEG
            .MODEL  small, pascal
            .STACK  ;4096

wordToDec   PROTO NEAR PASCAL decAddr:WORD, hexData:WORD

;-----------------------------------------------------------------------
; Macro   printst
;           This macro is used to print a string passed as an input
;           parameter and a word value immediately after the string.
;           The string is delared in the data segment routine during
```

```
;              assembly time.  The word is converted to dec ascii and
;             printed after the string.
;
; Input:    stringData = string to be printed.
;             wordData = word to be converted to dec ascii and printed
;
; Destroys: None
;
; Output:    None
;
; Assumes:  Stack is available
;
;-------------------------------------------------------------------------
printst MACRO     stringdata, hexWord
            local       stringlabel, decData

            .data

stringlabel         DB      stringdata
decData             DB      5 dup (0)
                    DB      0dh, 0ah, '$'

            .code

            pushf
            pusha

            ; Convert the word ino hex ascii and store in the string
            invoke  wordToDec, offset decData, hexWord

            mov     dx, offset stringlabel              ; Setup string to be printed
            mov     ah, 09h                             ; Execute DOS print function
            int     21h

            popa
            popf

ENDM


SEG_BIOS_DATA_AREA  EQU    40h
OFFSET_TICK_COUNT   EQU    6ch
INTERVAL_IN_TICKS   EQU    91


; Data segment

            .DATA


; Code segment

            .CODE
            .686p

cpufreq PROC NEAR
            local       tscLoDword:DWORD, \
                        tscHiDword:DWORD, \
                        mhz:WORD,\
                        Nearest66Mhz:WORD,\
```

```
                Nearest50Mhz:WORD,\
                delta66Mhz:WORD


        .startup                                ; Allow assembler to create code that
                                                ; initializes stack and data segment
                                                ; registers

; Step 1.

        ;Verify Genuine Intel processor by checking CPUID generated vendor ID

        mov     eax, 0
        cpuid

        cmp     ebx, 'uneG'                     ; Check VendorID = GenuineIntel
        jne     exit                            ; Jump if not Genuine Intel processor
        cmp     edx, 'Ieni'
        jne     exit
        cmp     ecx, 'letn'
        jne     exit

; Step 2 and 3

        ; Get CPU feature flags
        ; Verify TSC is supported

        mov     eax, 1
        cpuid
        bt      edx, 4t                         ; Flags Bit 4 is TSC support
        jnc     exit                            ; jump if TSC not supported

        push    SEG_BIOS_DATA_AREA
        pop     es
        mov     si, OFFSET_TICK_COUNT           ; The BIOS tick count updateds
        mov     ebx, DWORD PTR es:[si]          ; ~ 18.2 times per second.

wait_for_new_tick:
        cmp     ebx, DWORD PTR es:[si]          ; Wait for tick count change
        je      wait_for_new_tick

; Step 4
        ; **Timed interval starts**

        ; Read CPU time-stamp
        rdtsc                                   ; Read and save TSC immediately
        mov     tscLoDword, eax                 ; after a tick
        mov     tscHiDword, edx

        add     ebx, INTERVAL_IN_TICKS + 1      ; Set time delay value ticks.

wait_for_elapsed_ticks:
        cmp     ebx, DWORD PTR es:[si]          ; Have we hit the delay?
        jne     wait_for_elapsed_ticks

; Step 5
        ; **Time interval ends**

        ; Read CPU time-stamp immediatly after tick delay reached.
        rdtsc
```

```
; Step 6

        sub     eax, tscLoDword                 ; Calculate TSC delta from
        sbb     edx, tscHiDword                 ; beginning to end of interval

; Step 7
        ;
        ; 54945 = (1 / 18.2) * 1,000,000  This adjusts for MHz.
        ; 54945*INTERVAL_IN_TICKS adjusts for number of ticks in interval
        ;

        mov     ebx, 54945*INTERVAL_IN_TICKS
        div     ebx

        ; ax contains measured speed in MHz
        mov     mhz, ax

        ; Find nearest full/half multiple of 66/133 MHz
        xor     dx, dx
        mov     ax, mhz
        mov     bx, 3t
        mul     bx
        add     ax, 100t
        mov     bx, 200t
        div     bx
        mul     bx
        xor     dx, dx
        mov     bx, 3
        div     bx

        ; ax contains nearest full/half multiple of 66/100 MHz

        mov     Nearest66Mhz, ax
        sub     ax, mhz
        jge     delta66
        neg     ax                              ; ax = abs(ax)

delta66:
        ; ax contains delta between actual and nearest 66/133 multiple
        mov     Delta66Mhz, ax

        ; Find nearest full/half multiple of 100 MHz
        xor     dx, dx
        mov     ax, mhz
        add     ax, 25t
        mov     bx, 50t
        div     bx
        mul     bx

        ; ax contains nearest full/half multiple of 100 MHz

        mov     Nearest50Mhz, ax
        sub     ax, mhz
        jge     delta50
        neg     ax                              ; ax = abs(ax)

delta50:
        ; ax contains delta between actual and nearest 50/100 MHz multiple
```

```
        mov     bx, Nearest50Mhz
        cmp     ax, Delta66Mhz
        jb      useNearest50Mhz
        mov     bx, Nearest66Mhz

        ; Correction for 666 MHz (should be reported as 667 MHZ)
        cmp     bx, 666
        jne     correct666
        inc     bx
correct666:

useNearest50MHz:
        ; bx contains nearest full/half multiple of 66/100/133 MHz

        printst "Reported MHz = ~", bx
        printst "Measured MHz =  ", mhz                  ; print decimal value

exit:

        .exit                                           ; returns control to DOS

        ret

cpufreq  ENDP

;-------------------------------------------------------------------------
; Procedure         wordToDec
;       This routine will convert a word value into a 5 byte decimal
;       ascii string.
;
; Input:   decAddr  = address to 5 byte location for converted string
;                     (near address assumes DS as segment)
;       hexData  = word value to be converted to hex ascii
;
; Destroys: ax, bx, cx
;
; Output:           5 byte converted hex string
;
; Assumes:          Stack is available
;
;-------------------------------------------------------------------------

wordToDec PROC NEAR PUBLIC uses es,
                  decAddr:WORD, hexData:WORD

        pusha
        mov      di, decAddr
        push     @data
        pop      es                                     ; ES:DI -> 5-byte converted string

        mov      ax, hexData
        xor      dx, dx
        mov      bx, 10000t
        div      bx
        add      ax, 30h
        stosb

        mov      ax, dx
        xor      dx, dx
        mov      bx, 1000t
```

```
        div     bx
        add     ax, 30h
        stosb

        mov     ax, dx
        xor     dx, dx
        mov     bx, 100t
        div     bx
        add     ax, 30h
        stosb

        mov     ax, dx
        xor     dx, dx
        mov     bx, 10t
        div     bx
        add     ax, 30h
        stosb

        mov     ax, dx
        add     ax, 30h
        stosb

        popa
        ret

wordToDec       ENDP

        END
```

インテル株式会社
〒300-2635 茨城県つくば市東光台 5-6
http://www.intel.co.jp/

241618-029JA
JPN/0507/PDF/SE/AEG/YK/HN